Software Quality Management VI

Springer
London
Berlin
Heidelberg
New York
Barcelona
Budapest
Hong Kong
Milan
Paris
Santa Clara
Singapore
Tokyo

C. Hawkins, M. Ross and G. Staples (Eds)

Software Quality Management VI

Quality Improvement Issues

Organised by
The British Computer Society

BCS

Proceedings sponsored by

ORIGIN

Conference endorsed by

CEPIS

Springer

Chris Hawkins
BCS Quality SIG Chairman *and* Perspective Technica, Reading, UK

Margaret Ross
Southampton Institute *and* BCS Quality SIG, UK

Geoff Staples
BCS (Hampshire) *and* BCS Quality SIG, UK

The front cover design is taken from Bootstrap 3.0 - Software Process Assessment Methodology by A. Bicego, M. Khurana and P. Kuvaja, pp 31.

ISBN-13:978-1-85233-021-7

British Library Cataloguing in Publication Data
Software quality management
6: Quality improvement issues
1.Computer software - Quality control - Congresses
2.Software engineering - Management - Congresses
I.Hawkins, Chris, 1951- II.Ross, Margaret III.Staples, Geoff
005.3'0685
ISBN-13:978-1-85233-021-7 e-ISBN-13:978-1-4471-1303-4
DOI: 10.1007/978-1-4471-1303-4

Library of Congress Cataloging-in-Publication Data
A catalog record for this book is available from the Library of Congress

The use of registered names, trademarks etc. in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant laws and regulations and therefore free for general use.

The publisher makes no representation, express or implied, with regard to the accuracy of the information contained in this book and cannot accept any legal responsibility or liability for any errors or omissions that may be made.

Typesetting: Camera ready by contributors

34/3830-543210 Printed on acid-free paper

SIXTH INTERNATIONAL CONFERENCE ON SOFTWARE QUALITY MANAGEMENT

CONFERENCE CHAIRMAN

C. Hawkins

CONFERENCE DIRECTORS

M. Ross
G. Staples

INTERNATIONAL ADVISORY COMMITTEE

H. Basson
J. Brinkworth
M. Curtis
K. Daily
T. Daughtrey
J. Dolado Cosin
A. Dorling
D. Edgar-Nevill
B. Garner
E. Georgiadou
A. Grady
E. Gray
R. Green
T. Hall
S. Hill
B. Hirsh
R. Hunter
G. King
D. Kitson
P. Kuvaja
H. Leung
P. Linecar
C. Mackie
D. Maisey
M. Mullerburg
B. Myers
M. Pivka
T.P. Rout
V. Sivess
A. Symons
A. Torn
H. Wickberg
D. Wilson
H. Younessi
S. Zahran

PREFACE

The Quality Special Interest Group of the British Computer Society presents the edited proceedings of their sixth International Conference on Software Quality Management (SQM'98) held in April 1998 in Amsterdam.

The objective of this series of annual conferences is to promote international co-operation among those concerned with software quality and process improvement, by creating a greater understanding of software quality issues and by sharing current research and industrial experience.

The papers cover a broad spectrum of practical experience and research. The topic areas include process improvement, maintaining a quality management system, quality metrics, human factors, project management issues, software tools and approaches to systems development.

The organisers would like to thank Origin for their sponsorship of the proceedings. The editors are indebted to the members of the International Advisory Committee for their support and for refereeing the abstracts and the final papers, as well as to the authors who have contributed to the success of this conference.

The Editors.

CONTENTS

Section 1: Process Improvement

ASTERIX Introduction of a Process Improvement using Tools to Support System Testing
John Brinkworth and John Llewhellyn, (Anite Systems, UK) 3

Practical Process Improvement Meets DSP Software Development
T Kaikkonen, (Nokia Mobile Phones, Finland), M Oivo, (VTT Electronics, Finland), J Puputti, (Nokia Mobile Phones, Finland), and M Vierimaa, (VTT Electronics,Finland) 15

Bootstrap 3.0 - Software Process Assessment Methodology
A Bicego (Etnoteam, Italy), M Khurana (Southampton Institute, UK), and P Kuvaja (University of Oulu, Finland) 26

A Qualitative Approach to Organisational Goal Setting for Information Systems Development
Peter DC Bennetts, Stella Mills (Cheltenham and Gloucester College of Higher Education, UK) and Trevor Wood-Harper (University of Salford,UK) 38

Section 2: Quality in Software Development

Experience with Software Quality / Usability Metrics
Judy Lau (Shanable Pty Ltd, Australia) and David N Wilson (University of Technology, Sydney, Australia) 53

Improving Information Systems Development by Managing Project Risk
Peter Smart, (Price Waterhouse, UK) 70

Information Quality Supported By Meta Model
Tatjana Welzer and Ivan Rozman, (University of Maribor, Slovenia) 81

Quality and RAD: A Contradiction In Terms?
Paul Allen, (SELECT Software Tools Plc, UK) 89

Transient Quality Performance within the 'SPAN' of new Software Development Process and Technology Establishment
Nikos Lykouropoulos and Andreas Maniatis, (IMF Ltd, Greece) 106

Providing Support for the Integration of Structured Methods and Quality Management Systems
J Barrie Thompson, Helen M Edwards and Colin J Hardy, (University of Sunderland, UK) 108

Section 3: Object Oriented Issues

Can You Have It All?": Managing the Time and Budget Against Quality Issue in a Dynamic Business Object Architecture Development
KSY Hung, AJH Simons (University of Sheffield, UK) and A Rose (CAD Consultants, UK) 121

Object-Oriented Software Development with Reuse
M Milankovic-Atkinson and E Georgiadou, (University of North London, UK) 135

Integration of Strategic, Tactical and Technical OO Measurement
Marjan Hericko, Ivan Rozman, Matjaz B Juric and Romana Vajde Horvat, (University of Maribor, Slovenia) 147

Section 4: Software Tools

A Lotus Notes Implementation of a Workflow Automation Tool for ISO 9001 Certification
Yan Ching Ying and Keith CC Chan, (The Hong Kong Polytechnic University, Hong Kong) 161

Automatic Software Evaluation Based on Domain Knowledge and Formal Evaluation Specification
Boris I Cogan and Tamara O Matveeva, (Far Eastern Branch of the Russian Academy of Sciences, Russia) 173

Section 5: Quality Issues

Australian Software Developers Embrace QA Certification
Aileen P Cater-Steel and Edmond P Fitzgerald, (University of Southern Queensland, Australia) 187

Management Systems: The End of the Beginning?
Roderick H Macmillan, (British Telecom Labs, UK) 199

Can Quality Survive Amidst Adversity?
W Pat Burgess, (Burgess Consulting Inc, USA) 204

An Analysis of Obsolesence Risk in IT Systems
M Bradley (Rolls-Royce and Associates Ltd, UK), RJ Dawson (Loughborough University, UK) 209

Constructing Standards for Cross-Platform Operation
MA Heather (University of Northumbria at Newcastle, UK), BN Rossiter (Newcastle University, UK) 218

The Practical Use and Abuse of Software Metrics
Kawal Banga, (KSB Associates Ltd, UK) 230

A Quality Software Process for Rapid Application Development
Gerry Coleman (Centre for Software Engineering, Dublin, Ireland), Renaat Verbruggen (Dublin City University, Ireland) 241

An Integrated Model for Impact Analysis of Software Change
Henri Basson, (Laboratoire du Littoral, France) 260

Metrics for Estimation
T.R. Judge and N.S. Mistry (Parallax Solutions Ltd, UK) 270

Author Index 285

Section 1

Process Improvement

ASTERIX
Introduction of a Process Improvement using Tools to Support System Testing

John Brinkworth, John Llewhellyn
Anite Systems Quality Unit
Slough, UK
Acknowledgement : Paul Starkey

Abstract

Anite Systems Space and Defence Division is a mature organisation, certified to ISO 9001 standards and possesses considerable experience in the application of the European Space Agency (ESA) Software Engineering Standards to major software systems development. We have been selected by the European Commission to perform a Process Improvement Experiment (PIE) under its European Systems and Software Initiative (ESSI). The project is entitled ASTERIX - **A**utomated **S**oftware **T**esting for **E**nhanced **R**eliability **I**n E**x**ecution.

The project's objective is to determine how greater attention to system-level testing can improve software product quality whilst reducing overall costs as measured across the full software development lifecycle (including extended warranty periods). The project is scheduled to run from early 1997 to late 1998.

This paper is the project's "Interim Dissemination" deliverable providing details on project progress to-date. It highlights the benefits the project's results will bring to the Software Quality, Process Improvement, and Space Software communities.

Introduction

This paper discusses a project currently being performed by Anite Systems Space and Defence division. ASTERIX's objective is to determine how greater attention to system-level testing can improve software product quality whilst reducing overall costs as measured across the full development lifecycle (including extended warranty periods).

Anite Systems Space and Defence Division is a mature organisation, certified to ISO 9001 standards and possesses considerable experience in the application of European Space Agency (ESA) Software Engineering Standards to major software systems development. We have identified testing as the aspect of our software development process which can most significantly benefit from the application of new methods and tools. We believe that any process improvements which can be made in this area will be of direct benefit to a wide variety of organisations, whether currently applying the ESA Software Engineering Standards or otherwise.

C. Hawkins et al. (eds.), *Software Quality Management VI*

The resources, measured both in terms of cost and elapsed time, which have to be allocated under normal circumstances to system-level testing mean that although we believe we do a good job of testing there is potential to do even better.

We believe that this situation can be improved significantly only if a high level of automation can be brought to such testing. ASTERIX provides the resources needed to establish whether such an approach, in conjunction with strong higher management support, does lead to real benefits on a major representative project.

The baseline project chosen for this experiment is commercially confidential but we can say that it is being performed to tight budget and schedule constraints, and is capable of benefiting directly from the application of the proposed techniques.

ASTERIX is also analysing comparable projects previously performed by us using our current development standards. This allows us to derive objective figures for the benefits of adopting a thorough approach to automated system-level testing.

We see the results of ASTERIX as having a wide potential application, and we welcome the opportunity to contribute to an overall improvement in the approach of the European software industry in this area. We are therefore taking care to see that the ASTERIX results reach the widest possible audience - hence this paper for the SQM98 conference. This paper forms the "Interim Dissemination" deliverable of the project (part of our contract with the European Commission) and provides details on project progress to-date.

Anite Systems' Motivation for the PIE

Anite Systems Space & Defence Division produces bespoke systems and software systems for a wide range of major clients including the European Space Agency (ESA), the UK Ministry of Defence (MoD), the German Space Agency (DLR), the British National Space Centre (BNSC), Hitachi, and many others.

We have particular strengths in spacecraft control and mission planning systems, real-time spacecraft simulators, ground station control facilities, Earth Observation data processing, avionics and robotics applications.

We have identified testing as a process in our software development approach that needs to be made more efficient, and we wish to establish whether this can be overcome with the help of the use of tools for automated software testing. If the ASTERIX PIE demonstrates that our proposed approach is practical, we will use the experience as a basis for updating our Quality Management System so that all future divisional project work benefits accordingly.

Current Situation

Anite Systems Space & Defence Division is a mature software engineering organisation, which has been certified to AQAP 1/13 and then ISO 9001 standards for many years for Defence projects and, since the beginning of 1996, to ISO 9001 standards for all work performed at our Bristol (UK) offices.

The divisional Quality Management System is closely based upon the ESA Software Engineering Standards, also known as PSS-05, which has been used for Space-related project work since the introduction of the standards in the early 1980s. The standards are also extensively used in the wider software development community by organisations such as the UK Defence Evaluation Research Agency (DERA), CERN and GEC. The experience of the division in working to these standards has been fed back into the evolution of the standards during this time.

We know from our own experience and that of many other European software development organisations which operate in our market areas, that testing is always the weakest link in the practical application of the ESA Software Engineering Standards. It is often the case that a significant amount of development work has been performed before sufficient attention is given to the manner in which the software will be tested.

When the Project Manager is put under budget and schedule pressure it is normal that the effort allocated to testing is the easiest to reduce without immediate visible consequences. In the past our clients have often accepted this approach as there is usually sufficient time following delivery of the software system to solve most of the remaining problems before the system is required to support major mission events such as spacecraft operations. In return for accepting this reduced level of testing, our clients have been able to benefit by having an unusual amount of flexibility in changing their requirements for the software, often at quite a late stage in the development of the system.

These arrangements are, we believe, now becoming impossible to maintain in our market area. This is due to the increasing budgetary pressure and the rapid trend away from Fixed Unit Price (Time and Materials) work performed at client sites with significant client involvement, to Firm Fixed Price developments mostly performed remotely from the client.

To correct this situation we have started a process of education, both of our own staff (in particular our Project Managers) and of our clients, in the importance of allocating sufficient resources to testing. We are also looking to technological solutions which will allow us to make much more efficient use of the resources allocated to testing. We have identified the use of automated software test tools as the most promising technological approach; this has led us to establish the ASTERIX Process Improvement Experiment project.

Definition of the Experiment

The Baseline Project for the ASTERIX Process Improvement Experiment (PIE) is a major commercially confidential development. However we have not been able to apply as many resources as we might have liked to automate the testing on the project.

The ASTERIX PIE is therefore scoped to allocate sufficient resources to an automated testing approach that will allow us to determine the optimum level of testing required for major projects, and to establish the correct proportion of the budget to allocate to testing in future projects.

We believe that the automated testing approach will, for most of our future projects, be beneficial in providing our clients with improved quality systems and in reducing the overall development costs once the costs of repairing faults under warranty have been taken into account. The main aim of ASTERIX will be to establish whether this is the case.

ASTERIX runs for a period of 18 months, spanning the implementation and testing phases of the baseline project and the most crucial part of the subsequent warranty period.

The emphasis within ASTERIX is on providing comprehensive system and acceptance testing facilities, using a combination of the following techniques and categories of tools:

- Creation of the tests using an appropriate tool with extensive scripting facilities.
- Performance of the tests, including regression testing where required because of code changes as a result of failed tests. Regression testing will also be performed post-delivery when changes are made as a result of errors detected under warranty. Regression testing becomes feasible on a system of this nature (including complex interactive tasks) only when the tests can be repeated automatically.
- Test management supported by appropriate database tools. These typically provide standard test reporting and handling mechanisms (such as the Software Problem Report), and allow essential statistics on the progress of the tests to be made available.

This approach will be supported by a measurement programme, as discussed below.

Organisational and "people" issues are important in the success of ASTERIX. One of our wider objectives is to convince our staff that testing is as noble an art as design or coding. Whilst we are happy to have staff who have a specialist interest in testing, we prefer most of our staff to be able (and happy) to perform either an analysis/design/coding role or a testing role on any project. In common with most organisations, our developers currently prefer the former role. ASTERIX should help to rectify this situation by removing the drudgery from the testing process and to make clear the crucial role it can play in the success of a project.

Measurement of Results

ASTERIX allows us to perform many more system and acceptance tests than is normally possible. The benefits of adopting such an approach have to be quantifiable in terms of clear improvements in the quality of the product as delivered to the clients, and preferably in reducing the overall costs as measured across both the development and the warranty phases.

The measurement programme involves a significant level of investment in order to ensure that the data, which will be derived from a variety of sources, is statistically valid. The success of the ASTERIX PIE depends upon being able to make a clear statement on the quality and cost benefits of the approach adopted.

Measurable Targets

Integral to the PIE is the definition of measurable goals or targets. If the targets are met, then we will be able to recommend the adoption of the relevant techniques, but even if the targets are not met then we will have valuable information which will be of interest in the context of software process improvement.

We have identified the following specific targets for the ASTERIX PIE, related to the baseline project:

- The total number of Software Problem Reports (SPRs) received during the period covered by the PIE should be at least 50% less than the corresponding number on previous similar projects.
- The total cost savings (compared with previous similar projects) due to the lower number of SPRs handled should exceed the cost increase due to the additional system and acceptance testing performed.

These targets reflect the overall goal to demonstrate that the approach proposed will result in real gains both for the customer (through receiving a better quality product, allowing him to reduce operational costs) and for Anite (through reducing the overall lifecycle costs).

Comparison Projects

In addition to collecting data from the baseline project we need good quality data from recent projects in order to allow us to determine whether the ASTERIX experiment has allowed us to achieve our targets.

To allow meaningful comparisons we are considering major developments, with extended maintenance periods that are typically greater than one year in duration. Anite Systems Space & Defence Division has recently performed several such projects of similar scope to the baseline project. For all these projects we have access to project data. This will provide the basic raw material to allow us to evaluate the effects of the ASTERIX approach.

Cost/Benefit Analysis

In order to determine whether the approach to system testing embodied in the ASTERIX experiment should be adopted on future Anite Systems projects, we have to perform a cost/benefit analysis, where:

- **The costs** are the additional expenditure required on a new project to support the automated system testing approach as implemented on ASTERIX.
- **The benefits** are:
    - anticipated reduced costs during the warranty period;
    - higher quality in the delivered product.

The costs should in principle be easy to measure from the ASTERIX project, although we have to establish the recurrent costs, i.e. those costs that would be incurred by a successor project building upon the ASTERIX approach.

Measuring Improved Product Quality

In order to quantify the quality of a software product, we have established a framework (a Quality Model) that allows different characteristics of the software to be measured and combined into an overall quality figure. A suitable starting point for such a model is the ISO/IEC Standard 9126: *Information technology - Software product evaluation - Quality characteristics and guidelines for their use* (ISO/IEC 9126: 1991 (E)).

The ISO/IEC Quality Model is based upon the definition of software quality characteristics and subcharacteristics, as shown below in Figure 1.

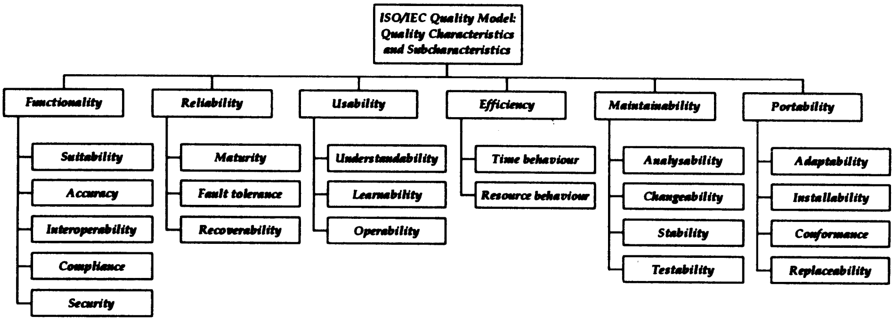


Figure 1

Note that the ISO/IEC Quality Model mandates the six basic quality characteristics to be considered in any specific quality model definition:

- functionality
- reliability
- usability
- efficiency
- maintainability
- portability

however the subcharacteristics shown in the figure above are purely indicative. The essence of the quality model is that it is possible to assess the software in terms of measurable quality (sub)characteristics, so that definition at least to the subcharacteristic level is required, even if a different set of subcharacteristics is identified.

Definition of a complete Quality Model, covering all the characteristics and sub-characteristics above, is beyond the scope of the ASTERIX experiment; many of the characteristics and sub-characteristics cannot be affected by the ASTERIX approach, and others (such as fault tolerance) are often not relevant for the types of software products we develop.

We have therefore concentrated on the following aspects of the Quality Model, as we perceive them to be directly relevant to the subject matter of the ASTERIX experiment:

- reliability (maturity only);
- maintainability (all sub-characteristics).

The act of defining each subcharacteristic within the Quality Model involves:

- providing a definition for the subcharacteristic itself (the definitions used within ISO/IEC 9126 are used);
- defining the specific attributes of the software which collectively constitute the subcharacteristic;
- defining a metric (i.e. a measurable characteristic) for each attribute;
- defining a means of converting from the measured value of each metric (which has a specific meaning for each metric) to a rating level which can have a common meaning across all metrics;
- defining the weightings which allow the individual metric rating levels to be combined into overall rating levels of:
    - the quality factors (which are the quantification of the quality characteristics);
    - the quality criteria (which are the quantification of the quality subcharacteristics).

Using these definitions, it is then possible to provide quantitative figures for the reliability and maintainability of the software, according to the following formulas:

$$F_i = \frac{1}{\sum_j \alpha_j}\left(\sum_j \alpha_j * C_j\right)$$

$$C_j = \frac{10}{3 * \sum_k \beta_k}\left(\sum_k \beta_k * M_k\right)$$

where:

F_i are the quality factors (normalised to lie within the range 0-10);

C_j are the quality criteria (normalised to lie within the range 0-10);

α_j are the weightings applied to the quality criteria when calculating the quality factors;

M_k are the assigned values of the rating levels of the metrics (in the range 0-3);

β_k are the weightings applied to the metric ratings when calculating the quality criteria.

Note that because F_i and C_j are normalised in these equations, only the relative values of the weightings within a given calculation are important, not their absolute values. For example, for a given quality factor, the following weighting schemes are identical in effect:

$\alpha_1 = 2;\ \alpha_2 = 1;\ \alpha_3 = 3$ and $\alpha_1 = 6;\ \alpha_2 = 3;\ \alpha_3 = 9$

The relationships between the metric ratings and the measured values of the metrics are specified individually for each metric. These relationships can be non-linear.

Following ISO/IEC 9126, we are using the following scheme for rating level definition:

- For any metric, it must be possible to map any possible value of the metric to one of four rating levels.
- The rating levels (i.e. possible values of M_k) are:
    - Excellent (assigned value 3);
    - Good (assigned value 2)
    - Fair (assigned value 1);
    - Poor (assigned value 0).

Any quality factors and criteria that are not relevant for ASTERIX purposes, or which will not be considered in ASTERIX, are assigned a weighting of zero.

As an illustration of the practical implementation of this approach, we provide below a simple example with the following features:

- only maturity is considered;
- the maturity is derived from only two metrics:
    - the number of new SPRs raised in each month;
    - the number of SPRs still open each month.

The mappings between the metric values and the rating levels is as follows:

New SPRs
Weighting: 4

Rating	From	To
3	0	9
2	10	24
1	25	49
0	50	

Open SPRs
Weighting: 7

Rating	From	To
3	0	24
2	25	49
1	50	74
0	75	

The simulated raw metric information (new SPRs and SPRs still open per month over a period of 12 months) is as follows in Figure 2:

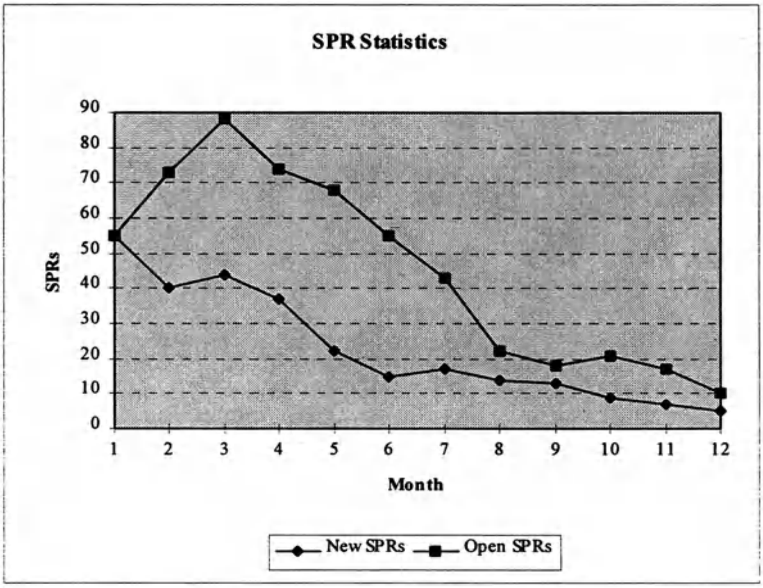


Figure 2

The calculated ratings, M_k, are as follows in Figure 3:

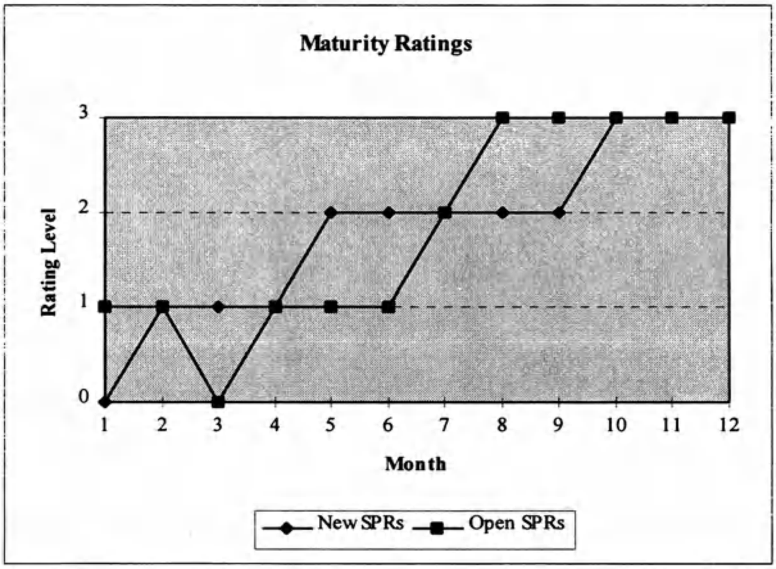


Figure 3

The resultant values, C_j, of the maturity are as follows in Figure 4:

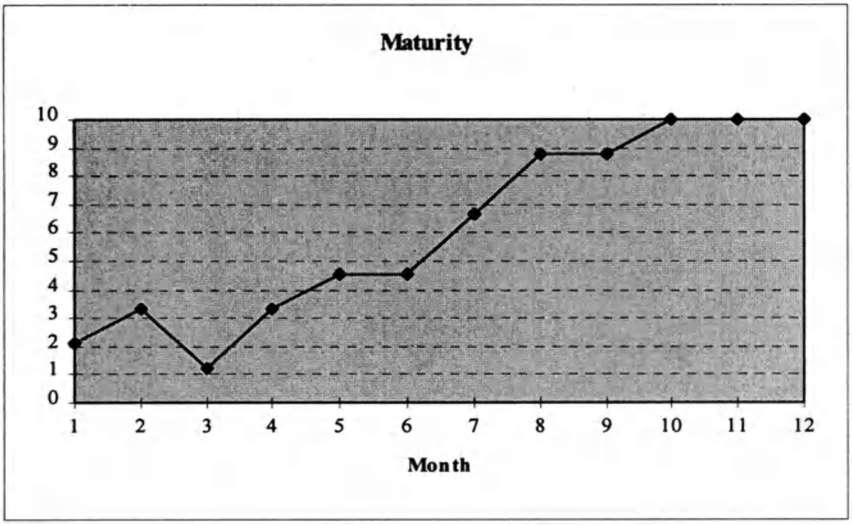


Figure 4

In practice the Quality Model produced by ASTERIX is more extensive than in this simple example, but the basic steps are the same. It is important to ensure that the mappings between the metric values and the rating levels are carefully thought out and are consistent across the developments under consideration (ie both the baseline and the comparison projects).

In this example the mapping would in general have to be based upon a measure of the size and/or complexity of the software (less than 10 new SPRs per month may be excellent for a large system, but poor for a much simpler system). We are considering a number of possible approaches to measuring the size of the systems and thus "normalising" the data. Potential candidate techniques include: number of lines of source code, number of specification pages, Function Points, cyclomatic complexity.

The following is the set of metrics that is being used for the subcharacteristics to be included in the Quality Model.

- **Reliability**
    - Maturity:
        - number of valid external SPRs accepted in the reporting period (assumed to be one month);
        - Percentage of all valid external SPRs that are still open;
- **Maintainability**
    - Analysability:
        - average effort to analyse and agree a solution to a problem reported by internal SPR;

  - average effort to analyse and agree a solution to a problem reported by external SPR;
- Changeability:
  - average effort to implement a solution to a problem reported by internal SPR;
  - average effort to implement a solution to a problem reported by external SPR;
- Testability:
  - average effort to test a solution to a problem reported by internal SPR;
  - average effort to test a solution to a problem reported by external SPR.

We do not intend needlessly to generate large numbers of metrics, rather the emphasis is on defining the minimum set of independent measures that fully capture the scope of the subcharacteristic.

Results so Far

Our results so far are limited since the project is not yet far enough advanced to provide conclusive figures on the experiment. Nonetheless the insights that have so far been uncovered are worth noting.

We have assembled the following "characteristic" data about the comparison projects.

Project	***Hardware***	***Operating System***	***Languages***	***Size in KLOC***	***Effort in effort-months***	***Total number of SPRs***
A	VAX ALPHA	VMS	C, Ada	22.8	122	694
B	SUN SPARC	SUNOS UNIX	FORTRAN, C++	TBD	120	TBD
C	VAX	VMS	FORTRAN, PASCAL	269	308	638
D	VAX	VMS	FORTRAN, C, UIL, PASCAL	888	334	857

Table 1

It should be emphasised that at present this data is tentative and will be subject to further validation.

The experiment is being performed in a commercial organisation. This introduces particular constraints and issues. Due to uncertainties in the baseline project we have had to in the first instance delay the testing activity, and subsequently transfer the experiment to another similar baseline project.

We have assembled a detailed scoring mechanism to assist with selecting the most appropriate tool to support the automated testing activity. We are now at the stage where the tool has been procured and is shortly to be applied to the next planned delivery for the baseline project.

The data gathering from previous projects has proved more troublesome than expected. We have developed all the systems using the same lifecycle methodology from the ESA standards however the search for data has exercised our filing, archiving and archive retrieval procedures more fully than is normally the case. We have assembled a fair amount of data but are aware of gaps. We expect to commence detailed analysis in the next few months.

There are organisational issues associated with introducing tools into a project whereby the fact that the tools are being introduced by a separate project from the baseline project can give rise to feelings of lack of ownership for the improvement process on the part of the development team. To attempt to address this issue a Service Level Agreement has been established between the PIE and the baseline project. This enables both partners in the experiment to understand their inter-relationship and obligations in terms of time commitments, tasks and deliverables.

We expect that our "final dissemination" paper will be able to provide a quantified assessment, using the techniques and models discussed above, of whether the process change was a process improvement.

Transferability of Experience and Potential for Replication

If we take the ESA Software Engineering Standards as a fair representation of "best practice" in the European software industry, then it is clear that the lessons learnt from ASTERIX can lead to direct improvements in these standards. We are in a particularly good position to influence these standards through our Quality Assurance Manager, Jon Fairclough, who is a co-author of these standards and a consultant to ESA for their development.

The ASTERIX PIE allows us to determine how best to use the testing resources at our disposal. It will demonstrate whether the injection of additional system and acceptance testing effort (made feasible by the use of automation techniques), can actually reduce the overall development costs as well as providing our clients with a better product.

The wider European software development community can benefit directly from the results of the ASTERIX work, in that the problem domain (system and acceptance testing) is common to all software developments. When completed ASTERIX should provide this community with figures showing the quantifiable benefits of adopting such an automated approach to testing.

Practical Process Improvement Meets DSP Software Development

T. Kaikkonen[a], M. Oivo[b], J. Puputti[a], M. Vierimaa[b]
[a] *Nokia Mobile Phones, PO Box 50, FIN-90571 Oulu, Finland*
[b] *VTT Electronics, PO Box 1100, FIN-90571 Oulu, Finland*

Abstract

The importance and effort invested in development of digital signal processing (DSP) software is increasing dramatically especially in telecommunication applications. Design of optimised mathematical algorithms has been dominating in research and product development, but as the size of software has grown, need for improved software development practices has emerged. Despite increasing importance, it's hard to find any reports on practical results of improving DSP software development processes. In this paper we report the experiences of DSP-ACTION - a joint-project of Nokia and VTT Electronics - which is carried out at the Mobile Phones division of Nokia corporation. Objective of the experiment is to demonstrate improvement of DSP software development process to support reuse of not only source code but every level of design documentation. The improvement programme is based on prior assessment, which was done by using Pr^{2}imer (Practical Process Improvement for embedded real-time software) framework. Measurements by using Goal/Question/Metrics (GQM)-approach are used to follow-up the improvement actions.

Keywords: process improvement, DSP, digital signal processing, GQM measurement

1 Introduction

The importance and effort invested in development of digital signal processing (DSP) software is increasing dramatically especially in telecommunication applications. Development of latest generation of mobile phones has increased the size of DSP software three to four times larger than the previous generation. DSP software operates in hard real-time environment with very tight time and performance requirements. Design of optimised mathematical algorithms has been dominating in research and product development, but as the size of software

C. Hawkins et al. (eds.), *Software Quality Management VI*

has grown, need for improved software development practices has emerged. Despite increasing importance, it's hard to find any reports on practical results of improving DSP software development processes.

In this paper we report the experiences of DSP-ACTION - a joint-project of Nokia and VTT Electronics - which is carried out at the Mobile Phones division of Nokia corporation. Objective of the experiment is to demonstrate improvement of DSP software development process to support reuse of not only source code but every level of design documentation. The improvement programme is based on prior assessment, which was done by using Pr^{2}imer (Practical Process Improvement for embedded real-time software) framework [1]. Measurements by using Goal/Question/Metrics (GQM)-approach [2] are used to follow-up the improvement actions.

The experiment was piloted in a real product development project aiming at solving practical problems in everyday design and development work in industry. Effective review practice was introduced by defining elements of DSP software quality after analysing the customer needs of both forthcoming and current projects. Set of documentation methods and tools were evaluated and selected so that they were compatible with current revision management system. It was hierarchically restructured to intuitively reflect every domain of pilot project's DSP software. By creating and collecting process documents into a single location, a base for continuous process improvement was founded. Expected impact of the experiment is faster cycles of high-quality product development with the use of collective base of design level experience. Quantitative measurements are used to guide current and future process development activities.

2 DSP software development

Digital signal processing (DSP) is the application of mathematical operations to discrete-time signals. Signals are represented digitally as sequences of samples which are obtained from the physical analog signals. DSP is central to the operation of modern telecommunication systems, such as modems or digital cellular telephones.

Currently dominant approach for DSP system design follows steps [3]:

1. System requirements definition (customer needs driven)
2. Signal analysis (data rates, parameters, noise sources, inputs and outputs)
3. Signal processing design (arithmetic analysis, graphs, architecture approach)
4. Resource analysis (hardware and software, memory utilisation)
5. Configuration analysis (partitioning according to step 4 and modes of operation)
6. Hardware and software design

A novel approach of DSP system development is hardware-software co-design. Similar design stages exists for both software and hardware development: requirements analysis, architecture design, specification, implementation, and testing. Every design stage produces documents which should be synchronised to certain versions of corresponding hardware and software design.

DSP software development process has become more and more important as the size of the software has grown. There are several elementary needs for improvement which may also exist in many areas where software development work has originally been hardware oriented but nowadays has become more and more software oriented.

2.1 Needs for DSP software development

DSP software operates in hard real-time environment with very tight timing, memory and performance requirements. Due to these requirements, DSP software requires highly optimised code which is mostly implemented using assembler language. Assembler language environment sets a challenge for the quality as assembler is a lower level language. For example, there are less tools and supportive software available for it than for example to C-language based environment. It is also more difficult for a novice designer to understand assembler. Good and clear commenting of the code during the implementation is important.

DSP algorithms are usually developed by researchers or system designers with quite high level mathematical models. DSP software designers seldom consider other solutions than simple line-by-line conversion from one assembly language to another. DSP designers have usually engineering background which emphasises hardware knowledge and mathematical algorithms. Therefore, DSP designers may have only little or no software engineering knowledge and even more, software processes and software methods are not widely known. This creates situation where "middle level" software architecture design is usually non-existent in DSP software design. In conclusion, training of software engineering skills is one of the important aspects when we want to raise quality of DSP software development.

DSP software development has evolutionary characteristics, software is rarely developed from scratch. Usually projects use code and documents from the earlier project and only a minor part of the code is completely new. Often all source code modules must be rewritten to a new assembly language variant for the new DSP processor. These kind of changes are dictated by hardware changes or performance requirements, and they do not add any new functionality to the existing code modules. Even though a lot of code is taken from old projects, code is not designed to be reusable, it is taken because it is known to work. Upper level documents, such as specification and design documents may be non-existent or they are not up-to-date. If reusability can be improved, it will lead to considerable savings in software development and shorten development time.

Usage of higher level languages, such as C or C++, are just emerging. They are traditionally been used for non-volume products, which run in workstations or a PC. However, as performance of DSP processors is increasing, price of memory is going down, and amount of software requirements exploding, the implementation language will be other than simple assembly also in volume products. For tight inner loops, assembly will remain as the only solution. This hybrid implementation approach sets new demands to software architecture design, implementation, and testing.

2.2 Needs for DSP software documentation

Development of DSP software as any other software is a process which produces documents. In order to support reusability, new projects must obtain all the necessary documents from the previous projects and these documents should be easily available and up-to-date.

In order to create necessary documents, documentation tools must be carefully selected and supported. There are a number of tools available for DSP development, and each of them has its own strengths and weaknesses. It is equally important that there are not too many tools for the same purpose. This may lead to problems because tools require support, document transfer from one tool to another can be difficult, and people must be trained to use tools.

Project documentation is often scattered and there is no explicit information available how to make certain documents. Also, there may not be many templates for documents. Clear work instructions, process descriptions and example documents make project documentation easier task.

Because of large number of documents and constant updates to documents, version control system is a necessity. It supports two needs for documents; first, documents can be easily located, and second, latest version of documents are found. This does not yet guarantee that documents are up-to-date, this can be achieved by quality guidelines and their implementation during the process.

3 Process improvement

This section describes the methods used in the DSP process improvement projects ACTION! and its follow-up project DSP-ACTION carried out at the Mobile Phones division of Nokia corporation. The purpose is to give an overview of the methods used.

3.1 Pr^{2}imer-method

Pr^{2}imer represents a practical and systematic approach to improving the quality of the software development process. It fully integrates software process analysis, modelling, improvement and measurement techniques in order to meet the critical quality requirements for the development of embedded systems. Pr^{2}imer addresses software process quality improvement by supporting the following tasks (figure 1): [1]

1. Quantitative and/or qualitative analysis of current software development practices
2. Definition of measurable goals for improving those practices
3. Planning of successive and practical process improvement steps
4. Piloting and trial use of the improved practices in product development projects.

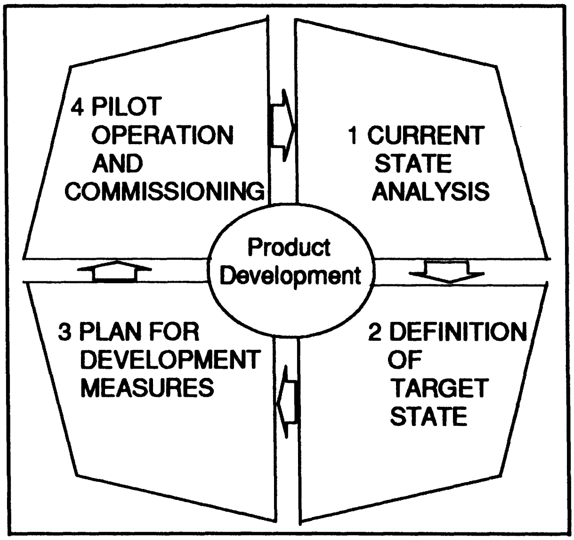

4 PILOT OPERATION AND COMMISSIONING
1 CURRENT STATE ANALYSIS
Product Development
3 PLAN FOR DEVELOPMENT MEASURES
2 DEFINITION OF TARGET STATE

Figure 1. Pr^{2}imer method [1]

First phase of the Pr^{2}imer method is current state analysis. It was carried out by interviewing several key persons of the DSP software development and using a questionnaire sent to DSP designers to every NMP's R&D sites which have DSP designers. Coverage of the interviews was 75% of projects (26% of designers) from the R&D site responsible for piloting improvement actions, but also engineering managers and other senior designers from 50% of R&D sites were interviewed. Approximately 40% of DSP designers answered to the questionnaire. As a result, analysis of the strengths and weaknesses of current process were known.

Definition of the target state is the second phase of the Pr^{2}imer method. In this phase, process improvement goals were set. The goals defined in this phase are measured using the Goal/Question/Metrics (GQM) software quality improvement paradigm [2].

In the third phase, software process improvement actions are taken step by step, concentrating on the areas for improvement identified during the analysis of the current software development process.

Pilot operation and commissioning is the fourth phase of the Pr^{2}imer method. Improved practices are piloted in real projects. If piloting indicates improvement, the revised practices can be institutionalised within the organisation.

In the ACTION!-project, improvement goals were identified according to the Pr^{2}imer method. One of the goals was to improve software documentation. Also, quality issues such as reusability were addressed when the goals were set.

Due to amount of work required, a separate project DSP-ACTION was launched. This project refined the original goals set during the ACTION!-project. Main goal of the DSP-ACTION project is to improve software documentation to promote reuse. Results of the DSP-ACTION project are measured using GQM approach.

3.2 GQM approach

Measurable goals of the DSP-ACTION project were defined in the beginning of the project. These were to "Analyse **the documentation process** in order to **improve it** with respect to **reusability**" and to "Analyse the **review process** in order to **improve it** with respect to **review effectiveness and quality**". After the goals were defined, we used GQM approach to refine selected goals into questions and metrics. Previous DSP software development projects did not use any kind of metrics to follow effectiveness of their processes.

GQM-approach uses structured interviews to capture the definitions, assumptions and implicit models related to the goal. The main purpose of the interviews is to make implicit knowledge of interviewed people explicit. Figure 2 shows the basic elements of the GQM approach.

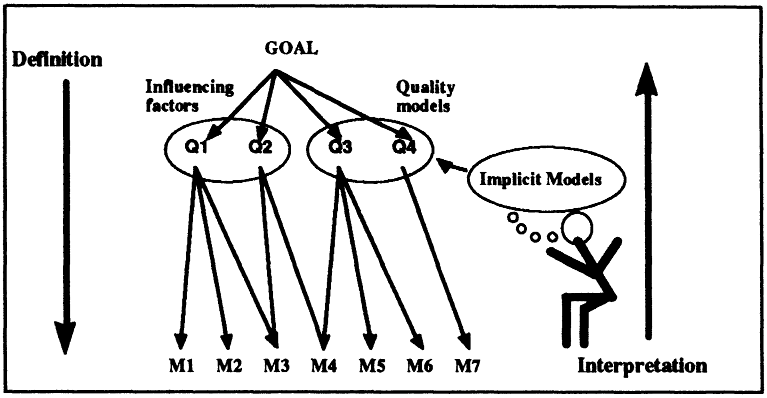


Figure 2 GQM approach

In interviews arranged during the DSP-ACTION project, we interviewed a total number of five people, both pilot project members and people who were responsible of the process improvement. Both GQM goals were addressed during the interviews, by asking the interviewed people their definitions and assumptions. A specific abstraction sheet was used to this purpose [6].

After the interviews, a GQM-plan was created which contained goals, questions and metrics. We created measurement plan that describes who will do the measurements, when they are done and how they are done. After the measurements are defined and pilot project has been selected, measurement program is started.

Feedback sessions are an essential part of the GQM approach, they are organised meetings involving pilot project members and GQM experts. Main objective is to discuss results from the measurements and obtain interpretation of measurements from the pilot project. Feedback sessions should be fully prepared and arranged regularly, at least once or twice between each major milestone on the project.

4 Results

Piloting baseline project is currently in its early stage. Therefore experiences can be estimated based on preliminary results of the measures obtained so far, but the final successes or failures of the overall project are yet to be seen. Significant amount of work has however been already done in the project. This work has provided significant improvement as described in the following paragraphs.

For average DSP designer, the process improvement experiment meant mainly three practical changes to the way work was done:

1. Every version of all kind of design documents were stored into PCMS[1]
2. Inspection was formally defined to consists of introduction and defined focus areas
3. GQM is used for measuring the status and turnout of the pilot project

Several tools that were considered suitable for DSP software development were evaluated by assessing their strengths and weaknesses from our point of view. Special attention was paid to the compatibility with version control system. Issues that were addressed included usability with PC/Windows and UNIX, support for large documents, support for algorithms and, current knowledge of tool. The evaluation of the tools was done in true working environment.

In order to define structures to the project documentation, all the documents (templates, work instructions, process descriptions) were collected from earlier projects and from document databases. Documents that did not have a template were taken as example documents for templates to be made in the future. Originally, we expected to concentrate largely to example documents and templates, but it was soon discovered that it was more useful to concentrate only to documents and templates that were really needed. If there were missing example documents or templates they are generated during the product development project work. Experience has proven that a good example really shows what should have been in the template in the first place. Also those who are experts on specific areas in DSP software development should also make the guidelines and templates how to create the documentation. Separate quality management team can not do this work for them, but can assist with the tools and general requirements.

[1] Process Configuration Management System by SQL Software

Quality and documentation plan was made to the pilot project. This document defined elements of quality, explained DSP software development process and supporting activities (for the process) such as reviews and inspections and, defined project documents and their exact locations.

Figure 3 shows how configuration management of DSP software development was done before and during the process improvement experiment (PIE). Previous projects had usually many problems in locating files, especially higher level design documents. In the new method, entire software was divided to logical algorithms, and their corresponding non-optimised implementation was placed near the same level with other documents. CMS[2] is configuration management location for early generation DSP software implementation. PCMS is currently used as a configuration management system. UNIX project directories were previously heavily used for storing (without version control) all kinds of design documents and specifications. COSSAP[3] is (non-version controlled) algorithm simulation environment, which C-model primitives were also stored in UNIX project directories. Test environment for unit testing was usually not version controlled, but stored in worst cases to personal home directories or even on PC. Now with the new PCMS hierarchy, all these can be version controlled and well managed. By generating hierarchical structure as defined in figure 3, we assume that it is easier to locate and manage project documents.

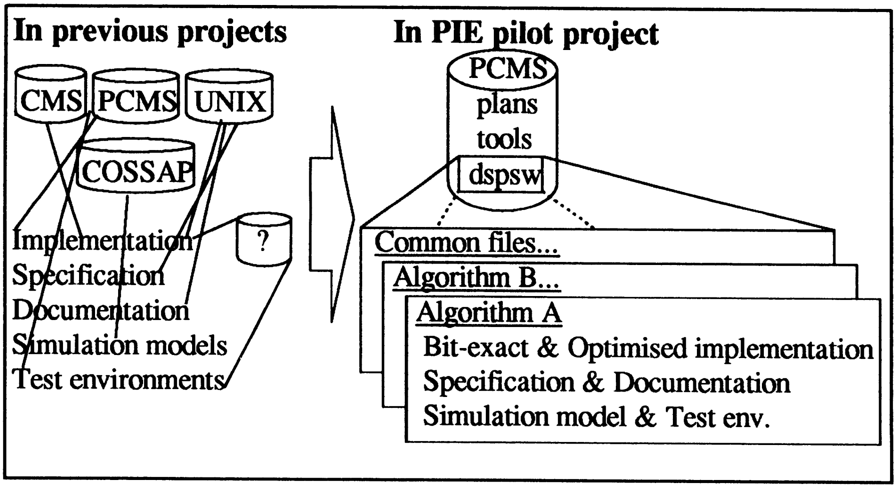


Figure 3: Hierarchical, revision controlled DSP software with documentation

Figure 4 shows how previous projects did their inspections merely ad hoc based on experience of each individual designer. This led to over-emphasis of hardware resource optimisations (memory, speed, power) based on reading assembly source

[2] Configuration Management System by Digital Equipment Corporation

[3] COSSAP DSP algorithm development environment by Synopsys

code line-by-line. Certain quality elements, such as testability or maintainability were often neglected. General methods [5] were studied to improve current inspection practices. The new inspection method consists of following stages:

1. Introduction to the topic (20-30 minutes), delivery of material and focus areas
2. Individual preparation to the inspection by reading from allocated viewpoints
3. Formal inspection (max. 2 hours) to collect all defects found from the reviewers

Previously no introduction was made and no focus areas defined. In pilot project, it is also required that for each algorithm at least following documents are prepared before the inspection:

1. Algorithm description
2. Source code modules for the actual implementation
3. Unit test environment with various test cases and expected results

For new algorithms, or those ones fundamentally improved, also specifications, simulation modes, and bit-exact non-optimised implementation area required to be inspected.

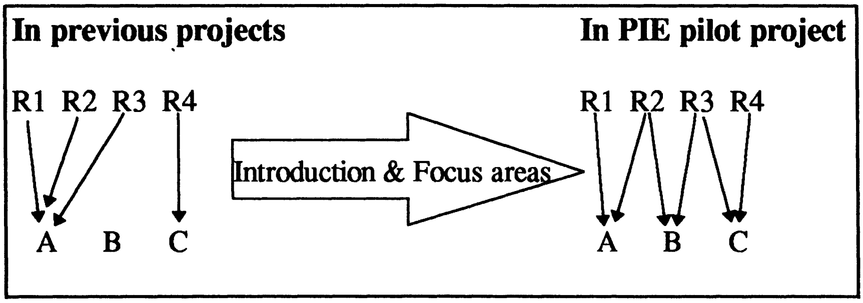


Figure 4: Quality focus oriented inspection practice

We defined quality of DSP software by the following key elements (figure 5), which were also used as focus areas for the inspection. All aspects of the quality must be considered while developing DSP software. In order to assure that no aspect has been overlooked. Specific checklists were defined to aid individual inspections to have consistent acceptance criteria.

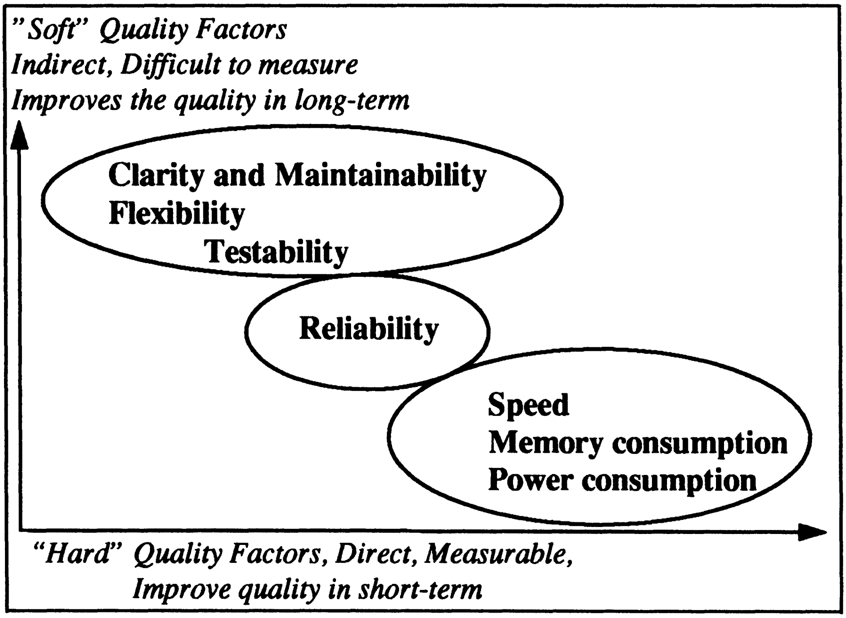

"Soft" Quality Factors
Indirect, Difficult to measure
Improves the quality in long-term
Clarity and Maintainability
Flexibility
Testability
Reliability
Speed
Memory consumption
Power consumption
"Hard" Quality Factors, Direct, Measurable,
Improve quality in short-term

Figure 5: Taxonomy of DSP software quality elements

Project group uses special Action Database (ADB) to store project reports, meeting minutes and information about projects defects and errors. Specific macros have already been created to obtain data directly from the database. These macros support automatic translation of data to Excel graphics. We modified this tool to support collection of all the required metrics thus making it possible to collect significant amount of metrics directly from the database, without any extra work from pilot project. This decreases effort for the metrics collection.

In order to make our improvements reality, training was needed. We trained the pilot project members, by explaining new solutions, methods and tools. Also, measurements that are collected to estimate project success were explained to pilot project. Pilot project was shown what information is required from them in order to collect the measurements.

We expect to measure quantitative results from reviews and reusability of the documents. Reusability measurements concentrate largely on existence of documents, whether a task produces all the necessary documents, whether those documents are up-to-date and so on. In form of measurements we expect to measure results, for example, from design such as percentage of design documents up-to-date, percentage of design documents done, percentage of design document reused from the old ones and so on.

From the inspections and reviews we measure how many defects were detected during the inspections, how many errors were not noticed during the inspections at certain stage, how long time it takes to correct a defect and so on.

There are not enough measurement data available yet from the pilot project so that conclusive numerical improvements could already be shown, but we will have more data as the pilot project matures. We expect to have some comparison data from the earlier projects, but as projects are always somewhat different, measurements are also compared to previous estimates of measured values.

5 Conclusions

In this paper we reported the experiences of DSP-ACTION - a joint-project of Nokia and VTT Electronics - which is carried out at the Mobile Phones division of Nokia corporation. Objective of the experiment was to demonstrate improvement of DSP software development process to support reuse of not only source code but every level of design documentation.

The work we have described has shown how the DSP software process can be improved using systematic approach and taking into consideration practical needs of the target project.

Acknowledgement

The work described in this paper was carried out with the support of the European Union under the ESSI process improvement experiment, project 23696, DSP-ACTION. In addition to the authors, individuals from VTT Electronics and Mobile Phones division of Nokia corporation have contributed to work reported here.

References

1. Karjalainen J., Mäkäräinen M., Komi-Sirviö S., Seppänen V. Practical process improvement for embedded real-time software, Quality Engineering, 1996, Vol. 8, no 4, pp. 565-573
2. Basili V., Caldiera G., Rombach D. Goal Question Metric Paradigm. In John J. Marciniak, editor, Encyclopedia of Software Engineering, volume 1, pp. 528-532. John Wiley & Sons, 1994
3. DeFatta D.J., Lucas J.G., Hodgkiss W.S. Digital Signal Processing: A System Design Approach, John Wiley & Sons, 1988, pp. 7
4. DSP Design Tools & Methodologies, The Buyer's Guide, Volume I, Berkeley Design Technologies Inc, 1995, pp. 61
5. Gilb T., Graham D. Software Inspection, Addison-Wesley Publishing Co, Reading, MA, 1993
6. Latum F., Solingen R., Oivo M., Hoisl B., Rombach D., Ruhe G. How to Adopt GQM-Based Measurement in an Industrial Environment, to appear in IEEE Software, January 1998

BOOTSTRAP 3.0 - SOFTWARE PROCESS ASSESSMENT METHODOLOGY

A Bicego
Etnoteam S.p.A., Milan, Italy
M Khurana
Systems Engineering Research Centre, Southampton Institute
Southampton, UK
P Kuvaja
Department of Information Processing Science, University of Oulu
Oulu, Finland
(in alphabetical order)

Abstract

BOOTSTRAP methodology was initially developed in an ESPRIT project together with European industry. After February 1993, the methodology has been managed and further developed by a European Economic Interest Group, called BOOTSTRAP Institute. BOOTSTRAP 3.0 was released in September 1997. It is compliant with the PDTR version of ISO 15504, the emerging standard on software engineering. The methodology contains a process model and an assessment method. The process model is based on the ISO 15504 reference model. In addition to the Process and Capability dimensions, it contains a Technology dimension. The Process dimension contains 33 different Processes organised in six clusters : Organisation, Life Cycle Dependent, Management, Support, Customer-Supplier, and Process Related. The Capability dimension consists of six levels, each level consisting of one of more Process Attributes, adopted from ISO 15504. An assessment is conducted at SPU and project levels. The BOOTSTRAP Institute organises and co-ordinates assessor training and registration scheme. BOOTSTRAP methodology is being used in two European projects, PROFES and SPAM.

1. Introduction

This paper introduces the latest version of the BOOTSTRAP software process assessment and improvement methodology. It describes the evolution of the BOOTSTRAP methodology as part of an ESPRIT project and the formation of the BOOTSTRAP Institute. Sections 4 and 5 describe the objectives and the main features of the methodology respectively. Finally, a brief description of the use of the methodology in European projects, PROFES and SPAM, is provided.

C. Hawkins et al. (eds.), *Software Quality Management VI*

2. Background

A European consortium partially funded by the European Commission within the ESPRIT program initially developed the BOOTSTRAP methodology. This project lasted from 1990 to 1992. The initial goal of the project was to fertilise the ground for good software engineering practices in Europe and to analyse the awareness of the European software industry in this area. The American experiences inspired the project to develop a kernel of a process assessment methodology based on the Capability Maturity Model (CMM) developed by the Software Engineering Institute (SEI) at Carnegie Mellon University. As at the time of the project, in Europe ISO 9000 standards were becoming the main reference also for the software industry; these standards were taken as major starting point to be integrated with the CMM approach. The newly developed approach was piloted during the initial development project, at Bosch GmbH Laboratories in Germany and it soon proved to be very helpful in establishing the basis for effective process improvement. Furthermore, the goal to provide a preliminary evaluation of the European awareness in software engineering provided the starting point for a database of assessment data. A more detailed explanation about the background and the previous versions of the BOOTSTRAP methodology are explained in [1].

After the completion of the ESPRIT project, some of the participating partners, decided to exploit its results by establishing an international organisation to professionally carry on the development and use of the methodology. This organisation is the BOOTSTRAP Institute.

3. BOOTSTRAP Institute

The BOOTSTRAP Institute was founded in March 1994 as a European Economic Interest Group (EEIG), a non-profit organisation legally established under the European law.

The objectives of the BOOTSTRAP Institute are to:

- Continuously improve the BOOTSTRAP methodology for the assessment and improvement of software process quality (also taking into account the relevant ISO standards and other international initiatives in the field including the proper packaging of the methodology and related training material);
- Promote the BOOTSTRAP methodology;
- License the BOOTSTRAP methodology to third Parties;
- Administer the database containing results of the assessments carried out by BOOTSTRAP licensees;
- Run courses for training and accreditation of BOOTSTRAP assessors;
- Certify the assessed organisations.

Within its objective of continuously enhancing the BOOTSTRAP methodology, the BOOTSTRAP Institute has developed a new release of the BOOTSTRAP methodology (Release 3.0). This is to ensure full conformance to the emerging ISO standard for software process assessment and improvement, better known as SPICE[1] (ISO 15504), and to align to ISO 12207 "Information Technology - Software Life Cycle Processes".

4. Objectives

The objectives of the BOOTSTRAP methodology are to:

- provide support to the evaluation of process capability against a set of recognised software engineering best practices;
- include internationally recognised software engineering standards as sources for identification of best practices;
- support the evaluation of how the reference standards have been implemented in the assessed organisation;
- assure the evaluation is reliable and repeatable;
- identify strengths and weaknesses in the organisation's processes;
- provide results that form a suitable and reliable basis for improvement planning;
- plan improvement actions that support achievement of the organisation's goals;
- help increasing process effectiveness while implementing standard requirements in the organisation.

5. Overview of the BOOTSTRAP Methodology

The main features of the BOOTSTRAP methodology include:

- the assessment method;
- the underlined process model;
- scoring, rating and result presentation mechanisms;
- process improvement guidelines.

5.1 The BOOTSTRAP Assessment Method

The BOOTSTRAP assessment is performed in three main phases:

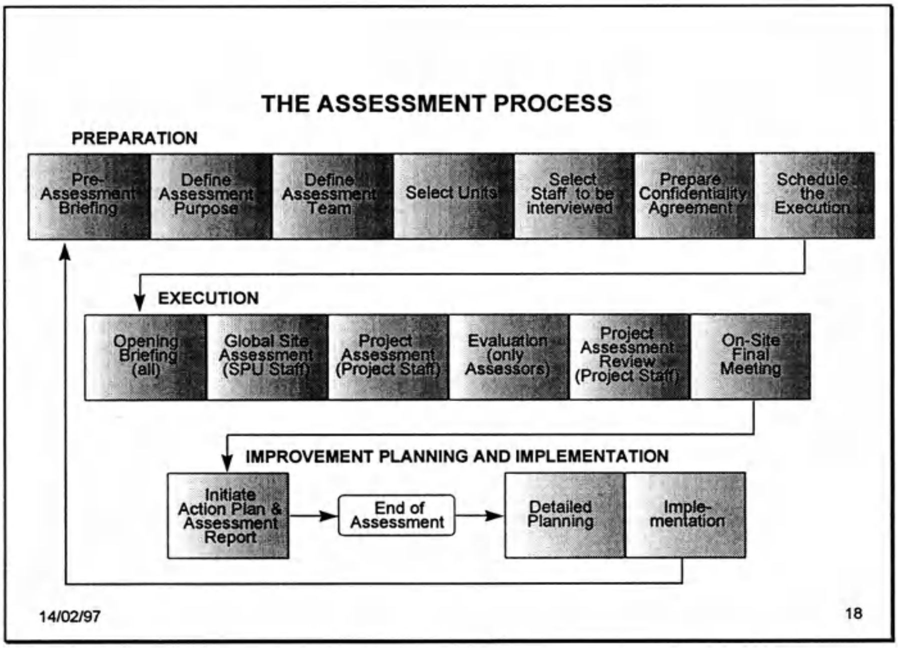
- The Preparation Phase, where context information are collected and the forthcoming steps are planned. This phase starts focusing on the organisation specific needs and objectives, which drive both the assessment as well as the improvement phase. Organisation needs and goals drive definition of the assessment scope, including processes where the assessment might need to be

[1] SPICE = Software Process Improvement and Capability Determination

focused, organisational units involved, documentation to be used, and key people to be interviewed. This phase includes an initial briefing to the organisation staff, with the purpose of increasing management commitment and creating awareness of issues related to the assessment and improvement.

- The Execution Phase, where information about the organisation's processes are collected by performing interviews of key personnel and evaluating available documents as planned in the Preparation phase. The information is collected at organisation level as well as at project level. Qualified assessors conduct interviews. Interviewees are always requested to support their statements with evidence. A preliminary presentation of assessment results completes the Execution phase. This is a review session aimed at correcting any misunderstanding that may have taken place while collecting information.
- The Improvement Planning Phase, where assessment results are used to identify suggested improvement areas and priorities. The consolidated assessment results and improvement recommendations are presented to the assessed organisation to get agreement and raise commitment for the subsequent improvement activities. The final output is an assessment report.

The three stages along with all the sub-stages of the assessment process are shown in Figure 1.



THE ASSESSMENT PROCESS

PREPARATION

Pre-Assessment Briefing

Define Assessment Purpose

Define Assessment Team

Select Units

Select Staff to be interviewed

Prepare Confidentiality Agreement

Schedule the Execution

EXECUTION

Opening Briefing (all)

Global Site Assessment (SPU Staff)

Project Assessment (Project Staff)

Evaluation (only Assessors)

Project Assessment Review (Project Staff)

On-Site Final Meeting

IMPROVEMENT PLANNING AND IMPLEMENTATION

Initiate Action Plan & Assessment Report

End of Assessment

Detailed Planning

Imple-mentation

14/02/97

18

Figure 1 BOOTSTRAP assessment process

5.2 The BOOTSTRAP Process Model

During a BOOTSTRAP assessment, the processes of the target organisation are evaluated to determine each process capability. Process capability is the ability of each process to achieve its goals in the context of the assessed organisation. Process capability evaluation is performed based on the BOOTSTRAP process model, which includes description of the processes and capability levels. The BOOTSTRAP process model is fully compatible with the requirements of the emerging international standard for software process assessment known ISO 15504 (PDTR version), being developed as part of the SPICE project.

The processes in the BOOTSTRAP process model form a tree structure (see Figure 2). Technology support is also addressed in the BOOTSTRAP evaluation. Technology assessment aims at evaluating the extent to which the process capability is strengthened with adoption of suitable tools for each process.

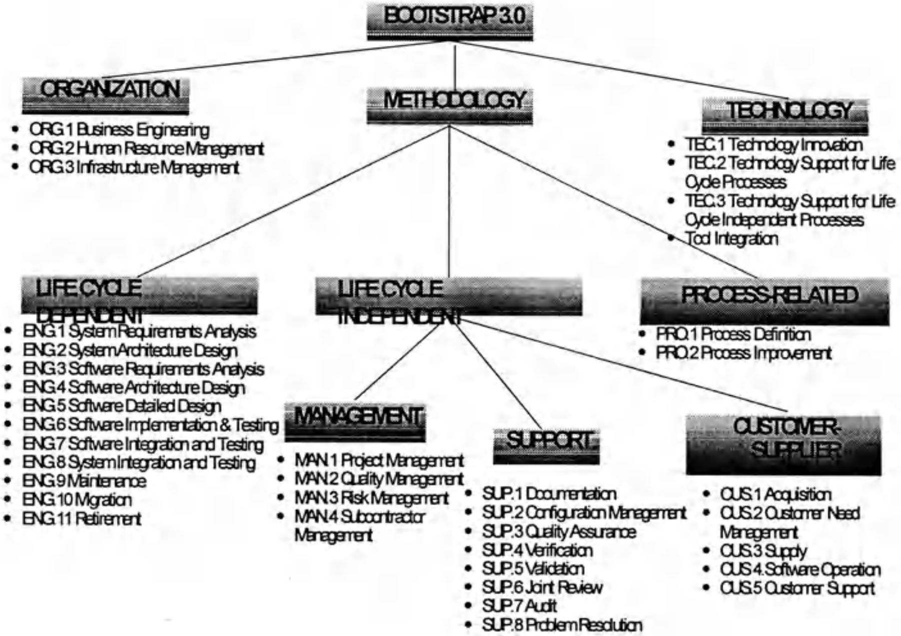

BOOTSTRAP 3.0
ORGANIZATION
• ORG.1 Business Engineering
• ORG.2 Human Resource Management
• ORG.3 Infrastructure Management
METHODOLOGY
TECHNOLOGY
• TEC.1 Technology Innovation
• TEC.2 Technology Support for Life Cycle Processes
• TEC.3 Technology Support for Life Cycle Independent Processes
• Tool Integration
LIFE CYCLE DEPENDENT
• ENG.1 System Requirements Analysis
• ENG.2 System Architecture Design
• ENG.3 Software Requirements Analysis
• ENG.4 Software Architecture Design
• ENG.5 Software Detailed Design
• ENG.6 Software Implementation & Testing
• ENG.7 Software Integration and Testing
• ENG.8 System Integration and Testing
• ENG.9 Maintenance
• ENG.10 Migration
• ENG.11 Retirement
LIFE CYCLE INDEPENDENT
PROCESS-RELATED
• PRO.1 Process Definition
• PRO.2 Process Improvement
MANAGEMENT
• MAN.1 Project Management
• MAN.2 Quality Management
• MAN.3 Risk Management
• MAN.4 Subcontractor Management
SUPPORT
• SUP.1 Documentation
• SUP.2 Configuration Management
• SUP.3 Quality Assurance
• SUP.4 Verification
• SUP.5 Validation
• SUP.6 Joint Review
• SUP.7 Audit
• SUP.8 Problem Resolution
CUSTOMER-SUPPLIER
• CUS.1 Acquisition
• CUS.2 Customer Need Management
• CUS.3 Supply
• CUS.4 Software Operation
• CUS.5 Customer Support

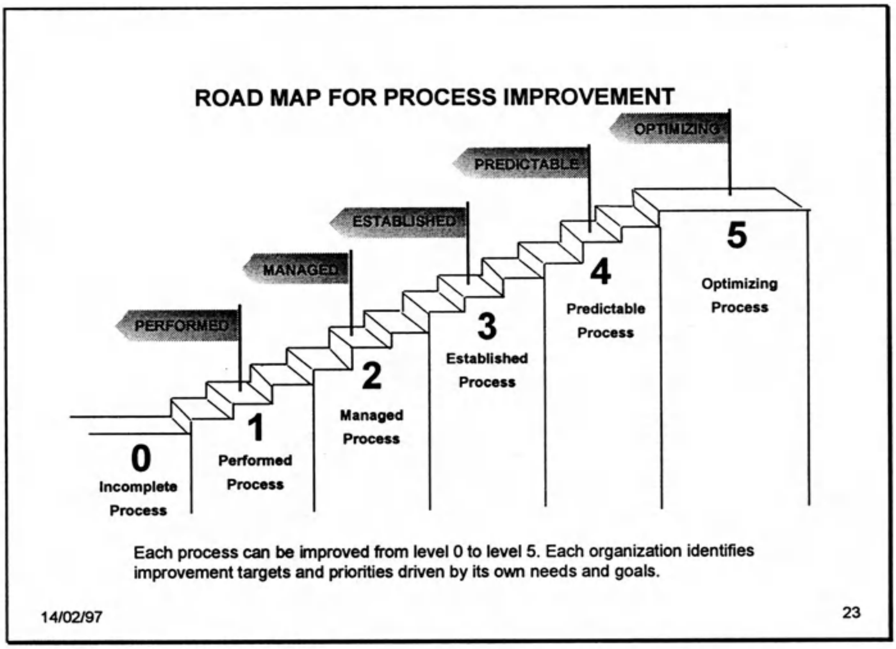
Figure 2 BOOTSTRAP Process tree

Process capability is measured based on the following capability levels (aligned with ISO 15504 requirements):

- Level 0: Incomplete Process - the process fails to achieve its purpose as it is incompletely implemented;
- Level 1: Performed Process - a set of practices is performed that allow the process to achieve its purpose;

- Level 2: Managed Process - the process delivers work products of acceptable quality within defined time-scale and resources;
- Level 3: Established Process - the process is performed based on a recognised organisational process definition;
- Level 4: Predictable Process - the established process is performed using defined quantitative control limits;
- Level 5: - Optimising Process - changes to the definition, management and performance of the process are identified and performed in a controlled way to continuously improve process performance.

The above capability levels apply to each process, resulting in a potential improvement path from Level 0 to Level 5, as shown in Figure 3.



ROAD MAP FOR PROCESS IMPROVEMENT

Each process can be improved from level 0 to level 5. Each organization identifies improvement targets and priorities driven by its own needs and goals.

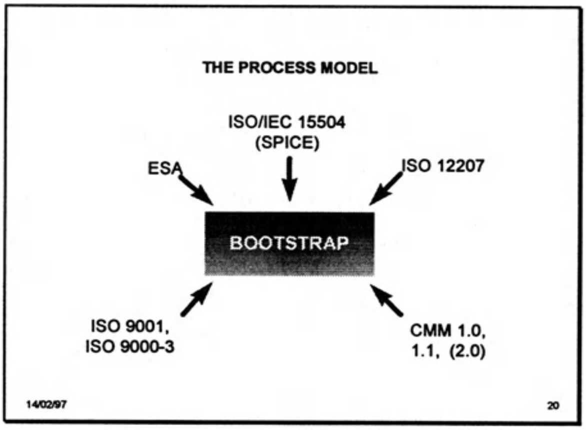
14/02/97

23

Figure 3 Improvement road map

The BOOTSTRAP process model integrates requirements from several internationally recognised standards like ISO 9001, ISO 9000-3, ISO 122207, ISO 15504, ESA PSS-05-0 and the CMM, as shown in Figure 4. The BOOTSTRAP process model contains cross references to the requirements of these standards, thus allowing evaluation of the assessed organisation against the selected reference standard, while evaluating process capability. Within the above mentioned standards the following ones play a particular role:

- ISO 9001 provides the organisational focus, particularly with regards to the quality system and organisation wide procedures;
- ISO 12207 provides a recognised framework for software processes;
- ISO 15504 provides the framework for process and capability definition;
- CMM has been one of the main references for the initial version of the BOOTSTRAP methodology and is intended to be used as a reference for further evolution of the methodology



THE PROCESS MODEL

ISO/IEC 15504
(SPICE)

ESA

ISO 12207

BOOTSTRAP

ISO 9001,
ISO 9000-3

CMM 1.0,
1.1, (2.0)

14/02/97

20

Figure 4. Integration of requirements of the international standards

5.3 Scoring, Rating and Result Presentation

Assessment results are the basis for process improvement planning. Effective process improvement can take place only if assessment data are reliable and provide a fair representation of the assessed organisation's capability. Reliability and repeatability are obtained by:

- ensuring that assessors have the required software engineering background and use the same approach (this is guaranteed by the BOOTSTRAP assessor accreditation scheme), and
- applying precise scoring and rating rules.

Each practice is evaluated based on a four-point scale. The capability level is then counted in two ways:

- by applying the BOOTSTRAP algorithm showing quartiles within each level;
- by applying the SPICE rules for deriving the capability level rating (where the overall results depend largely on the assessor's evaluation).

Final assessment results are presented as capability profiles of each process assessed (see Figure 5). These output profiles are specific to the BOOTSTRAP methodology.

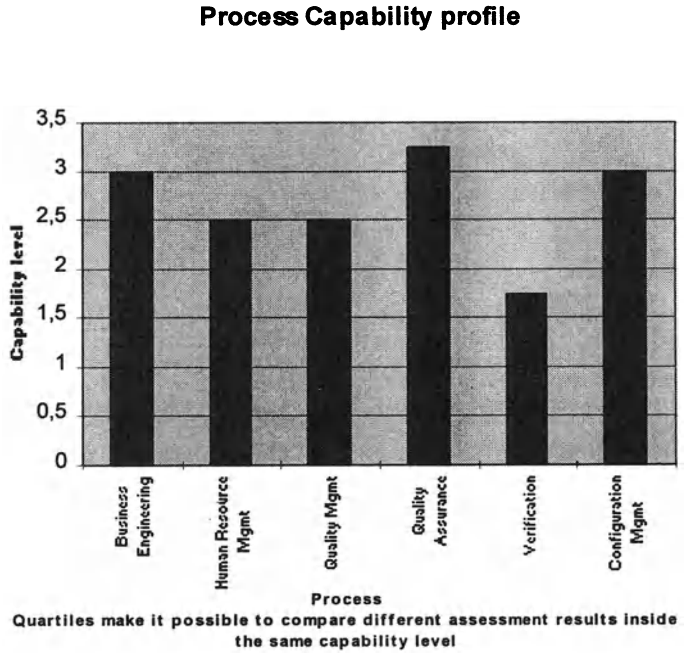


Figure 5. Capability profile of each process assessed

The capability profile is produced at two levels:

- the SPU[2] level, and
- the project level.

The SPU level profile shows the capability of organisational processes and reflects the management's point of view. The project level profile shows the process capability of the individual projects. At least two projects must be assessed to perform a complete BOOTSTRAP assessment. It is also possible to focus the assessment on a subset of processes, particularly to validate the results of improvement efforts. SPU results are not a mere integration of project profiles but

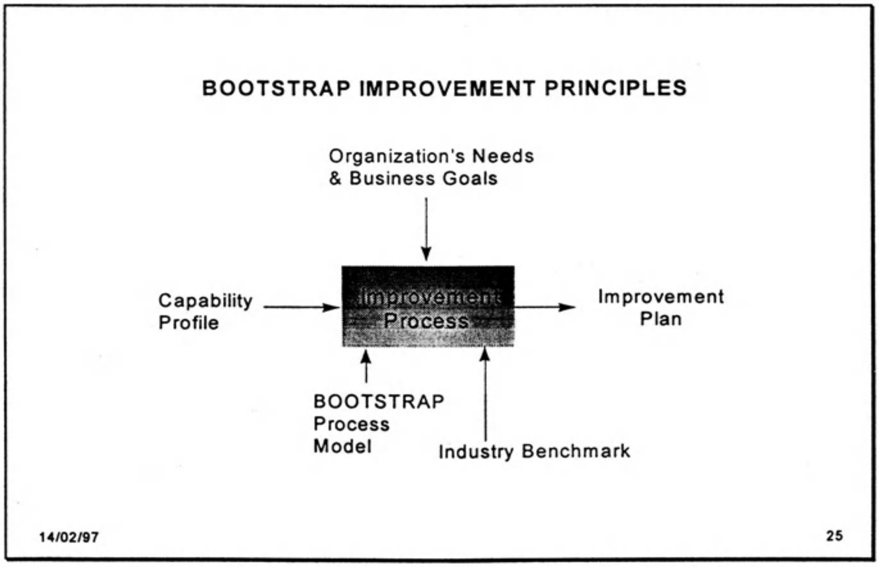
[2] SPU = Software Producing Unit

show a distinct set of findings. Comparison between SPU and project profiles provides invaluable information to support improvement planning.

In addition to capability profiles, the BOOTSTRAP assessors collect information about the current practices, problems and needs as perceived both by practitioners and managers. This information is used in improvement planning.

5.4 Process Improvement Guidelines

A common issue when implementing software-engineering standards in an organisation is how to tailor the standard's requirements to the organisation's needs and goals. Sometimes standard requirements seem to negatively affect achievement of the organisation's goals, in that they mandate rules that might reduce flexibility and timely answers to external stimuli. It is difficult to find a unique way to implement software engineering standards and to identify a common improvement path suitable to all kinds of organisations. The BOOTSTRAP methodology considers organisation needs and goals, capability profile of it's processes, and industry benchmarks as the main drivers for process improvement as shown in Figure 6. The BOOTSTRAP process model, aligned to ISO 15504, provides an improvement path for each process, but does not provide any suggestions on how to prioritise process improvement. Defining priorities is up to each organisation. The BOOTSTRAP methodology provides guidelines to identify which processes highly affect achievement of the organisation goals; then improvement priorities are assigned to each process. Processes with low capability but high impact on the organisation's goals are prioritised the highest.



BOOTSTRAP IMPROVEMENT PRINCIPLES

Organization's Needs
& Business Goals

Capability Profile

Improvement Process

Improvement Plan

BOOTSTRAP Process Model

Industry Benchmark

14/02/97

25

Figure 6 BOOTSTRAP improvement principles

As part of the improvement plan mitigation actions are identified to account for the risks, should the identified improvement actions fail to achieve their purpose. Typical risks include insufficient management commitment, insufficient resources, and barriers to accept change.

6. BOOTSTRAP methodology in use in other projects

The BOOTSTRAP methodology is currently being used in two projects in Europe, namely PROFES and SPAM.

6.1 PROFES

An ESPRIT project called "PROFES" (PROduct Focused improvement of Embedded Software processes) started in January 1997 with the aim of developing a customer oriented product driven improvement approach for embedded systems. The intention is to link the product characteristics directly to the process characteristics and enable a continuous product improvement. This is being done by integrating SPICE conformant (in this case BOOTSTRAP) approach with the most recent Goal Question Metric (GQM) methodology and process modelling. One of the deliverables of the project will be an enhanced version of BOOTSTRAP methodology validated in three industrial application experiments.

The project is in its first phase. Already, experiments carried in three industrial partners show that product viewpoint needs more visibility is software process assessments. The viewpoint is only indirectly addressed in while assessing relationship of customer with it's supplier. The enhanced version of the BOOTSTRAP methodology will be verified in the second phase of the project beginning in April 98.

6.2 SPAM

The primary objective of SPAM (Software Portability Assessment Methodology) is to develop a methodology for assessing software portability.

SPAM's underlying philosophy is that adequate solutions to software portability cannot be obtained unless both process and software aspects are considered (as with many other aspects of software quality). Therefore the objective of the SPAM methodology is to provide a comprehensive and repeatable approach for assessing how portable a software product is and how geared towards portability is the development process that produced it. This is achieved by taking into account four main aspects: the development process, the usage of the programming language, Application Programming Interfaces conformance and actual platform dependencies.

The process model for portability assessment is based on Bootstrap 3.0 and includes the process model itself, the process indicators for portability, and guidelines to conduct process assessment for portability.

7. Conclusions

BOOTSTRAP methodology was developed as part of an ESPRIT project especially by taking the European industry requirements into account. This brought ISO 9001 conformance into BOOTSTRAP. Another clear advantage was to show the capabilities of single processes as capability profiles and with quartile precision. It is being further developed by the BOOTSTRAP Institute by enhancing especially the risk management features and assesment process into the methodology. BOOTSTRAP Assesor training and acreditation has produced about 300 trained BOOTSTRAP Asessors mainly into Europe. To date there are also about 50 registered BOOTSTRAP Lead Assesors.

The methodology includes a process model, an assessment process, a mechanism to score the process and present the results, and guidelines on process improvement. The current version of the BOOTSTRAP Methodology is conformant with the emerging ISO 15504 standard of software engineering.

The BOOTSTRAP methodology is being used in two European projects - PROFES and SPAM. In PROFES, the methodology is being integrated with the GQM approach to produce an improvement methodology for embedded systems. In SPAM, the process model for portability assessment is based on the Bootstrap methodology.

Acknowledgement

This paper was written in an ongoing Esprit project PROFES, and is regarded as a PROFES related paper.

References

[1] Kuvaja, P., and Bicego, A., BOOTSTRAP: Europe's assessment method, IEEE Software, 1993, Vol. 10, Nr. 3, pp. 93-95

[2] Kuvaja, P., Similä, J., Krzanik, L., Bicego, A., Koch, G., and Saukkonen, S., Software Process Assessment and Improvement. The BOOTSTRAP Approach, Blackwell Business, Oxford, UK, and Cambridge, MA 1994.

[3] Humphrey, W. S., Managing the software process. Addison-Wesley Publishing Company Inc., Reading, Mass., 1989.

[4] Paulk, M., et al. Capability Maturity Model for Software, Version 1.1, CMU/SEI-93-TR-24, Feb. 1993.

[5] Paulk, M., et al. Key Practices of the Capability Maturity Model, Version 1.1, CMU/SEI-93-TR-25, Feb. 1993.

[6] ISO 9001. Quality Systems. Model for Quality Assurance in Design/Development, Production, Installation and Servicing. International Organisation for Standardisation, Geneva, 1989.

[7] ISO 9000-3. Quality management and quality assurance standards. International Standard. Part 3: Guidelines for the Application of ISO 9001 to the Development, Supply and Maintenance of Software,ISO, 1991.

[8] ISO 9004-4. Quality management and quality system elements. International Standard. Part 4: Guidelines for quality improvement, ISO, 1993.

[9] Paulk, M.C., and Konrad, M.D., An overview of ISO's SPICE project. American Programmer, February 1994.

[10] SPICE - Software process capability determination standard product specification for a software process capability determination standard. Document WG10/N016. ISO/IES JTC1/SC7/WG10, 1993.

[11] ESA Software Engineering Standards ESA PSS-05-0. Issue 2. ESA Board for Software Standardisation and Control, European Space Agency, Paris, February 1991.

[12] Bicego, F. et al., *Profes : Announcing The Marriage Between Process Assessment And Measurement*, International Conference of Software Engineering ICSE'97, Boston, 1997

Section 2

Quality in Software Development

A Qualitative Approach to Organisational Goal Setting for Information Systems Development

Peter D.C. Bennetts, Stella Mills
Department of Information Technology
Cheltenham and Gloucester College of Higher Education

and
Trevor Wood-Harper
Department of Mathematics and Computer Science
University of Salford

Abstract

The rational approach to information systems development has been recommended by many. However, this approach ignores problems associated with individuals and the organisational context, as it sees them as irrelevant and this can cause problems with the users' perception of the relevance of the system. The systems approach is seen as an appropriate way to address these issues. Furthermore, the rational approach assumes that the organisation will set clear and appropriate goals with which to monitor the quality of the software produced. Organisations appear to find this quite difficult to perform. One response to address this has been Basili's Goal / Question / Metric approach. This is defined in a way which requires quantitative measurements to be taken through appropriate metrics. If a systems approach is used, the concerns addressed are likely to be qualitative or interpretative in nature. This paper shows that by adapting the definition of a good (quantitative) metric appropriately by loosening the criteria slightly, qualitative metrics can be approved. Subsequently, Basili's Goal / Question / Metric approach can be adapted by using this revised approach to metrics, to support information systems development within a systems approach. A theoretical example is developed in the context of a case study based on a large organisation in the financial sector.

C. Hawkins et al. (eds.), *Software Quality Management VI*

1 Introduction

Information systems development usually has to satisfy three groups of people - the managers, the developers and the users. Historically, developers have usually made every effort to resolve technical problems implied by a specification. Unfortunately, it appears that implemented software often fails to satisfy users because non-technical issues, such as user views, politics or organisational context have not been taken sufficient note of. Consequently, although advances have been made using a purely rational approach, systems still fail. Other issues which may cause problems for information systems include goal setting and the measurement of goal achievement. In order to address these issues an interpretative approach is required. There are, of course, classes of situations where the traditional, rational approach is usually successful. For example, if clear and unambiguous goals can be set for a project and these goals conform to the organisation's goals. This approach advocates quantitative measurement and for software quality this is achieved through the use of metrics. This paper will give an overview of what makes a good metric. Unfortunately, goals are not always easy to define. One approach to address this has been developed by Basili [1] - the Goal / Question / Metric (GQM) approach. However, both the concept of a good metric and Basili's GQM approach need to be adapted to be usable in an interpretative context. These changes are identified and a theoretical case study developed within an existing organisation.

2 Background

It is clear, from the literature, that there can be problems of various kinds when developing software and that these have been recognised for many years. For example, Sauer [2] and Smith and Wood [3] show that adequate control is not always being realised. Further, Pressman [4] and Scach [5] consider that the delivered code often fails to achieve its objectives. These are two problem areas which give rise to the perception of a crisis of confidence that suggests that it is very difficult to provide the required software features on time and to budget. This crisis of confidence has been given the title 'The Software Crisis'. Current thinking sees this problem as endemic and although improvements have occurred, the crisis is not likely to be eliminated [5,6].

In 1968, a Study Group on Computer Science, established by the NATO Science Committee, responding to the perception of a software crisis (which was causing concern even then), recommended the holding of a working conference on Software Engineering [7]. The term 'software engineering' was coined to be deliberately provocative, implying, as it does, the need for software development to be based on the principles and practices seen in engineering [7]. For Boehm [8], 'Software engineering is the application of science and mathematics by which the capabilities of computer equipment are made useful to man via computer programs, procedures and associated documentation.'

The NATO Study Group's belief that the use of applied science would be an appropriate solution to the software crisis was echoed by Hoare [9] when he said that 'professional practice ... [should be] ... based on a sound understanding of the underlying mathematical theories and ... should follow closely the traditions of engineers in better established disciplines'. Gibbs [10] still supports this view when he concludes that a disaster 'will become an increasingly common and disruptive part of software development unless programming takes on more of the characteristics of an engineering discipline rooted firmly in science and mathematics.' This is the same recipe as reported by [7] nearly twenty years earlier. From the above description we can characterise the software engineering perspective as deterministic and one that assumes that there is some definable, true and real set of requirements that can be elicited and formally specified. It concentrates on the production of a piece of software that conforms to a specification as efficiently as possible [11].

However, Lucas [12] asserts, based on his empirical studies, that 'technical quality is a necessary, but not sufficient condition for successful systems. Technical quality can not be defined solely by the criteria of the computer scientist, but rather the functions of the system as seen by the user determine technical quality'.

Having characterised the software engineering approach and recognised that it is not itself sufficient for success, it is worth identifying the perceived reasons for failure in order to identify issues which are not addressed by the rational approach. Lyytinen [13] examines the failures found in development and in use separately. He claims that the top three problems in development involve goals, process features and self-image; while in use, the three most common issues were complexity, concept and people's reactions. Similar problems are identified by Ashley [14]. The traditional approach assumes that goals are clear, unambiguous and non-conflicting. However, the reality of most organisations is that change is not as rational as theory would demand. Further, organisational goals do not necessarily agree with informal objectives. Also, many stakeholders do not identify themselves with these goals. The rational approach 'neglects' the techniques of politics required to attain consensus. For example, one particular technique to achieve consensus is to make the goal sufficiently ambiguous.

Newman [15] puts it quite strongly when he suggests that it is a myth that organisational issues are not the concern of its information systems professionals. Miles [16] supports this when he argues that analysts tend to ignore the political or social dimension. Further, Lucas [17] argues that most information systems problems relate to 'organizational behavior problems' with these systems. Lucas [17] and Symons [18] recognise how political the development of an information system can be. For example, a perceived benefit for one group is not necessarily a benefit for another group as increased access to information in one place may imply an eroded power base elsewhere [18]. Markus [19] makes it quite explicit when she says 'information systems do not exist in isolation: they have the potential for creating major changes in power relationships and stimulating conflict.' Similarly, 'no one knows how many computer-based applications ... are abandoned or expensively overhauled because they were unenthusiastically received by their intended users' [19]. We argue, therefore, that organisational politics is a key issue. Indeed, Lyytinen [20] reports on research by Vitalari which concluded that there was a positive correlation between the interests analysts took in the organisational politics related to a system and the rating the analysts received from the users. However, Symons [18] considers that despite often playing a key part in ISD, politics '... are generally ignored by researchers'.

While a common response to the software crisis has been to try to improve 'best practice', practitioners and managers do not always do what they know they should. For example, Gibbs [20] reports Larry E. Druffel, Director of Carnegie Mellon University's Software Engineering Institute as saying that unfortunately, 'the industry does not uniformly apply that which is well-known best practice'. Even if 'best practice' is enshrined in a methodology, this does not guarantee success. For example, a discussant (Eddie Moores) at the 1995 Information Systems Methodologies Conference, referring to some currently unpublished research of his, said that the most used methodology was 'JDI' (Just Do It). It was also made clear in subsequent discussion, that even when managers said a methodology was being used, this was no guarantee that this was so. This is also confirmed by a case study described by Playford [21].

A focus on formality and automation, as recommended by software engineers, will only make things better if the people problems have been addressed beforehand. As Pirsig [22] notes, a quality product will only be produced if people care about quality. Formality and technology will not, in themselves, improve the situation. The politics and expertise has to be right first. This is confirmed by Playford [21], when reporting a case study that he had undertaken. This addressed the problem as to why the Structured Systems Analysis and Design Method (SSADM) was considered a failure in a particular organisation. The main factors were seen to be:-

- Poor and very limited training at an operational level;
- Limited training for team, sectional and departmental management;
- Isolation of pilot projects during installation and initial running;
- Isolated instances of associated skills training;
- Limited and short-sighted departmental training programme;
- Inappropriate applications for initial projects;
- Over-dependency on the mechanistic aspects of the methodology.

Hence, we deduce that although SSADM may well give good results in a supportive environment, the environment will not be affected by following the processes of SSADM alone and that in that non-supportive environment projects are likely to be unsuccessful.

Recently, it has been pointed out, by Hirschheim and Klein [23], that it is important to recognise the way analysts are seen by the organisation and how they see themselves. This can be identified by the use of metaphor. The politically unaware analyst or software engineer will be seen as a technocrat who tries to deliver what is thought of as the user's requirements, but with minimal reference to the user. The non-technical reasons for failure are more likely to be addressed when the analyst acts as a facilitator. This change of emphasis is actually quite radical, as it reflects a major change in the background philosophy of the analyst. For the facilitator, the approach is interpretative and contextual in outlook, it views models as a means of talking *about* reality rather than models *of* reality. This new philosophy cannot be supported entirely from within the current scientific framework. There are, therefore, certain limitations to the science paradigm when considering individual members of staff and managers. These are addressed by a related paradigm - the systems approach or systems theory ([24, 25, 26]; for example). Briefly, a systems approach will reflect a concern to look at potential systems as a whole (holistically). This is in contrast to the scientific or engineering approach which tends to solve problems by breaking them down into smaller, more manageable fragments (reductionism). Thus the holistic approach will include subjective issues, whereas the scientific or engineering approach attempts to be purely objective. Hence, it has been argued that approaches to ISD should not be purely scientific, as there are important issues for ISD which would otherwise be ignored. Software quality assurance (SQA) can be considered to be a monitoring and control process over ISD. The change of viewpoint identified above will affect what is meant by software quality and the way that it is measured.

3 Software Quality

Software quality is very difficult to define. Indeed work by Bennetts and Wood-Harper [27] indicates that different people have different implicit characterisations of it. Pirsig [22] shows that, at the very least, software quality contains subjective elements and consequently works against a purely objective approach to ISD. This conclusion is supported by Dromey [28] when he comments that 'notions - like 'quality', 'goodness' and 'fitness-for-purpose' - are *experiential*' (original emphasis). This is important as, if one of the goals for an ISD project were 'Develop a high quality product', the identification of achievement is necessarily subjective.

The rationale for the measurement of software quality is often characterised by quotations such as:

> 'to measure is to know' J.C. Maxwell (1831 - 1879)
>
> 'When you can measure what you are speaking about and express it in numbers, you know something about it, but when you cannot measure it, when you cannot express it in numbers, your knowledge is of a meagre and unsatisfactory kind' Lord Kelvin (1824 - 1907)

Both these quotations reflect the confidence of physical science as it was towards the end of the last century, when Newtonian physics seemed to show the way of solving all recognised problems. Indeed, Lord Kelvin is also well known for a quotation from a talk he gave to a group of erstwhile physics students, when he suggested that they should change their area of interest, as very soon all the problems of physics were expected to be solved. The thrust towards formality and measurement as a response to the software crisis echoes that confidence. However, this paper has been at pains to emphasise that the problems of ISD in an organisation are not just one of physics, logic and technology.

Despite earlier comments, there are good reasons for attempting to measure a project team's achievements. Basili [1] points out that the processes of software development need feedback and evaluation in order for the process to be both controlled and better understood. A similar point is made by Wallmüller [29], when he claims that measuring supports objective evaluation of performance, emphasises problems to management and clarifies objectives to engineers. Fenton [30] makes similar comments.

The measures that are used in software quality are known as metrics. Given the difficulties raised above, it is worth summarising the main reasons for using metrics:

- Target setting - In order to be able to track targets and goals, they must be set in a measurable way [30].
- Project planning and prediction - The prediction and estimation of required resources, costs and time scales can be enhanced by comparing the new project with similar, earlier projects [31, 32]. Progress statements can be quantified.
- Project management - The project manager needs metrics data to monitor and control the project [30, 31].
- Quality improvement - process and product - Metrics can be used to measure processes as well as the product. Miller [33] recognises that the overall quality of a product can only be improved by improving the development process as well.
- Productivity improvements - Similarly, improvements to productivity may be achieved through metrics [31].
- Customer confidence - The use of a programme of improvement to enhance software quality should impress the potential user [31].

For Watts [34] and Gillies [35] the criteria for a good metric are:-

- Objective - measurement not influenced by the measurer.
- Reliable - results should be precise and repeatable.
- Valid - the metric must measure the correct characteristic.
- Standardised - the metric is unambiguous and allows comparisons.
- Comparable - the metric must be comparable with other measures of the same criteria.
- Economic - the simpler (and hence cheaper) a measure the better.
- Useful - the measure must address a need not just measure a property for its own sake.

Wallmüller [29] gives a similar set of criteria.

However, it was argued earlier that successful ISD uses a systems paradigm which will incorporate interpretative issues and values. It is therefore useful to extend the concept of metric in a corresponding manner, by relaxing the criteria on objectivity and reliability. Hence a metric is said to be 'pseudo-objective' if the metric measurement is either independent of the measurer or reflects a consensus among all stakeholders. Similarly, a metric is said to be 'pseudo-reliable' if, at each invocation the stakeholders arrive at an unambiguous value for each instance. No further amendments seem to be necessary. The metric, even though qualitative and evaluated through consensus, still needs to be valid, standardised, comparable, economic and useful. In order to reflect this change, two new terms are coined. Metrics defined according to the original criteria should now be known as 'hard' metrics. Metrics defined as suggested, to include subjective or interpretative values, should be known as 'soft' metrics. The word 'metric', with no associated adjective, will refer to both hard and soft metrics.

4 Defining Measurable Goals

It is recognised that measurement is needed in order to monitor and control software quality. Many metrics programmes have failed because the objectives of the exercise were not fully identified [36]. Basili and colleagues have developed a 'rigorous' approach to remedy this called the Goal / Question / Metric (GQM) approach. Basili and colleagues offer a template which requires managers to resolve problems of purpose, perspective and environment in order to define a goal [1]. Each goal generates questions which are quantifiable and which are themselves answerable through the evaluation of identified metrics. The metrics may be scalar or a distribution. Further, the same questions may refer to more than one goal [1]. Within the template, Basili recognises a need to 'define the object or objects of study, what we are going to do and why we are doing it' [1]; this is the 'purpose' of the goal. It is also recognised that a particular viewpoint (or more than one viewpoint), of whoever is to receive information concerning the goal, needs to be defined so that relevant information is formulated. This would seem to recognise an interpretative approach. The environment has to be identified in order to clarify the 'context of the study' [1]. This identification will need the relevant factors to be identified. The idea is that these factors should be consistent across several goals within the project.

Basili tackles product-related objects and process-related objects slightly differently. Products and processes are characterised by defining the product or process itself; defining the quality viewpoints of interest and generating relevant feedback. These activities differ only in the way the product or process is defined. A product is defined by its logical and physical attributes, its cost, changes and defects and its context. All these items are expected to be quantitative. On the other hand, a process is defined by its process conformance ('a quantitative characterisation of the process and an assessment of how well it is performed') and domain understanding ('a quantitative characterisation of the object to which the process is applied and an analysis if the process performer's knowledge concerning this object and its domain') [1]. By addressing these issues simple models of products and processes are developed. It is noted, however, that Basili [1] offers an example operational mode; whose characteristics are qualitative (i.e. ordinal) rather than quantitative. Again, an example questionnaire supplied in an appendix to Basili [1] has many questions which are qualitative or interpretative. However, recently there has been a significant move from the scientific / engineering domain to the interpretative, as rehearsed earlier. This point is important because Basili's description of his approach refers to quantitative results throughout, notwithstanding the use of qualitative questions, as above.

The assumption here is that goals will be defined and then evaluated through their characterising questions and identifying metrics. Given the emphasis on quantifiable values, it is further assumed that, ideally at least, these metrics satisfy the criteria given earlier for a good (hard) metric. The first two criteria for a good metric were that the measurement was not influenced by the measurer - it was Objective; and that the results should be precise and repeatable - i.e. Reliable [35]. These criteria are unlikely to be satisfied by the results of a subjective enquiry. Basili's starting point is the rationalist paradigm, which earlier argument showed was not entirely adequate. The subsequent argument showed that this paradigm should be extended to and replaced by a systems paradigm. This would enable interpretative issues to be included. It is recognised that Basili needs to incorporate these issues, but his epistemological and ontological assumptions should exclude them. Consequently, Basili's metrics are recognised as being 'hard' in the sense introduced earlier. However, GQM should be extended to include 'soft' metrics, as what is being attempted by Basili should appropriately include these. In addition, using 'soft' metrics will support the advocated processes.

When discussing how his approach may be operationalised, Basili [1] himself talks about '... providing a set of ordinal values associated with the various steps of the process ...'. However, this does not reflect the assumptions identified earlier. On the other hand, this does imply not only that Basili's operationalisation of the approach does not need to be changed but also that subjective measurements have already been used with this approach in practice. Hence, Basili's Quality Improvement Paradigm (ie the Project Organisation (which performs the project characterisation, goal setting, process choosing and execution steps) and the Experience Factory (which performs a *post mortem* analysis on the data and information gathered and hence suggests appropriate amendments to processes and models)) will be essentially unchanged by introducing the concept of soft metrics. However, there still remain the apparent inconsistencies of Basili's work mentioned above. A small case study is offered to confirm these ideas.

5 The Case Study

This case study concerns a large company in the financial services sector with branches across the UK. They have a leading edge culture in terms of client-server technology. Their software has to operate on a wide range of platforms, including mainframes. They use, or expect to use soon, Information Engineering with CASE and configuration management tools among other elements of software support.

Quality as seen by the customer, is recognised by post-implementation faults, which are examined and backtracked to design. A customer satisfaction survey system exists, but is not yet used by senior management. For this organisation, Fagan inspections are achieved using walkthroughs, but the cleanroom regime is thought too bureaucratic. The software development process is controlled via quality manuals and reflecting customised systems life-cycles. Risk management is seen as a relative weakness within the organisation as it is not yet felt to be good enough. The risks for each project are defined by the project leader and embedded into each project's quality plan. Users are consulted throughout development. The organisation is looking for facilitators able to use a development tool set, recognise group dynamics and ensure that even reticent or inexperienced users have every opportunity to contribute to the specification and design process.

Several organisational goals have been identified which relate to the production of software. However, for the purposes of this paper, one particular goal will be used as an example - 'Focus on the customer and give them what they require'. The GQM approach [1] requires that appropriate questions are generated to characterise the identified goals. One such question is taken from Basili's Appendix 2 - 'What is the importance of testing each requirement?'. This question will also be used as a soft metric whose range of values is identified by the following scale:-
0 - not important, could be left out
1 - not too important, may affect some users
2 - mildly important, will affect some users
3 - important, should affect most users
4 - extremely important, part of the essence of the system
5 - critical, without this the system is useless
x - do not know.
For a particular project, each requirement can be evaluated for importance by a representative committee of stakeholders to develop a corporate view.

The characterisation of a good soft metric suggests the following analysis for this metric. The values given to the metric reflect a consensus of the stakeholders, by assumption, and so the metric can be said to be 'pseudo-objective'. Similarly, the stakeholder committee may not come to the same decisions if they repeat the exercise, as the political environment will have changed; so the metric is 'pseudo-reliable'. The metric gives a scale of the requirements' perceived importance and so measures the correct characteristic. It is unambiguous, provided a consensus is arrived at and will allow comparisons to be made. It may not be very economic, given the investment of effort required. There is certainly no question of any qualitative metric being completely open to automatic evaluation in the way McCabe's complexity metric is [29]. However, the soft metric described above, is useful as it allows rational decisions to be made about project management and scheduling. It is therefore argued that this metric can be classified as a good soft metric. Further, it is concluded that the identification of soft metrics can be useful as it regularises qualitative measurements.

6 Conclusions

This paper has shown that qualitative goals need to be identified for ISD. In order to achieve this, Basili's GQM approach has been adapted, principally by extending the notion of a good metric. In order to maintain the current concept of a good quantitative metric, the term 'hard metric' has been introduced. This is distinguished from a good qualitative metric (or 'soft metric') by relaxing the two principal characteristics of a metric (objectivity and reliability). The revised characterisations were identified. A glimpse of how soft metrics might work in practice was provided by the case study. Hence, this revised characterisation of metrics will allow qualitative issues to be evaluated. Consequently, Basili's description of his approach can be adapted so that 'quantitative' can be replaced by 'quantitative or qualitative' and hence qualitative goals identified.

References

1. Basili, Victor R. Applying the Goal / Question / Metric Paradigm in the Experience Factory, In Fenton, Norman, Whitty, Robin, and Iizuka, Yoshinori (eds.); Software Quality Assurance and Measurement A Worldwide Perspective; 1995 (International Thomson Computer Press, London) Chapter 2, p.23-44.

2. Sauer, Chris Why Information Systems Fail: A Case Study Approach 1993 (Alfred Waller Limited, Henley-on-Thames).

3. Smith, David J. and Wood, Kenneth B. Engineering Quality Software (2nd edition) 1989 (Elsevier Applied Science, Barking, Essex).

4. Pressman, Roger S. Software Engineering - A Practitioner's Approach (2nd edition) 1987 (McGraw-Hill).

5. Scach, Stephen R. Classical and Object-Oriented Software Engineering (3rd edition) 1996 (Irwin, Chicago).

6. Sommerville, Ian Software Engineering (4th edition) 1997 (Addison-Wesley Publishing Company).

7. Naur, Peter, Randell, Brian and Buxton, J.N. (eds.) Software Engineering - Concepts and Techniques 1976 (Petrocelli / Charter, New York).

8. Boehm, Barry W. Software Engineering Economics 1981 (Prentice-Hall Inc., Englewood Cliffs, NJ).

9. Hoare, C.A.R. Programming is an Engineering Profession Technical Monograph PRG-27 May 1982 (Oxford University Computing Laboratory, Programming Research Group).

10. Gibbs, W. Wayt, Software's Chronic Crisis, Scientific American; September 1994, p. 72-81.

11. Vidgen, Richard, Wood-Harper, Trevor and Wood, Robert (J.R.G.), A Soft Systems Approach to Information Systems Quality, Scandinavian Journal of Information Systems 1993 Vol. 5, p. 97-112.

12. Lucas, Henry C.Jr. Implementation The Key to Successful Information Systems 1981 (Columbia University Press, New York).

13. Lyytinen, K., Different Perspectives on Information Systems: Problems and Solutions ACM Computing Surveys 1987; Vol. 19., No. 1, p. 5-46.

14. Ashley, Nicholas, Measurement as a Management Tool, In Kelly, Mike (ed.); Management and Measurement of Software Quality 1993 (Avebury Technical, Cambridge) Chapter 4, p. 31-49.

15. Newman, M., Some Fallacies in Information Systems Development, Int. J. of Information Management 1989 Vol. 9, p. 127-143.

16. Miles, R.K., Computer Systems Analysis: The Constraint of the Hard Systems Paradigm J. of Appl. Sys. Anal. 1985 Vol. 12, p. 55-65.

17. Lucas, Henry C.Jr., Why Information Systems Fail, 1975 (Columbia University Press, New York).

18. Symons, Veronica, Evaluation of information systems: IS development in the Processing Company, J. of Information Technology 1990 Vol. 5, p. 194-204.

19. Markus, M. Lynne, Power, Politics and MIS Implementation; Communications of the ACM 1983 Vol. 26, No. 6, p. 430-444.

20. Lyytinen, Kalle, Stakeholders, Information System Failures and Soft Systems Methodology: An Assessment J. of Appl. Sys. Anal. 1988 Vol. 15, p. 61-81.

21. Playford, Michael Andrew, Failure to Thrive: A Study of the Implementation of Methodologies in Bureaucratic Data Processing Environments 1997 M.Phil Thesis, Anglia Polytechnic University.

22. Pirsig, Robert M., Zen and the art of Motorcycle Maintenance, 1974, (William Morrow & Co., New York).

23. Hirschheim, Rudy and Klein, Heinz K., Four Paradigms of Information Systems Development, Communications of the ACM, 1989, Vol. 32, No. 10(Oct), p. 1199-1216.

24. Checkland, P.B., Systems Thinking, Systems Practice, 1981, (John Wiley & Sons, Chichester).

25. Lyytinen, Kalle and Hirschheim, Rudy, Information systems failures - a survey and classification of the empirical literature, Oxford Surveys in Information Technology, 1987, Vol. 4, p. 257-309.

26. Weinberg, Gerald M., Rethinking Systems Analysis and Design, 1988, (Dorset House Publishing, New York).

27. Bennetts P.D.C. and Wood-Harper, A.T., Multiple Viewpoints and Software Quality Goals, In Samson, B., Marshall, I.M., Edgar-Nevill, D.G.(eds.), Proceedings of 5th Software Quality Conference, 1996, p. 66 - 75.

28. Dromey, R. Geoff, Cornering the Chimera, IEEE Software, 1996, Vol. 13, Part 1 (January), p. 33-43.

29. Wallmüller, Ernest, Software Quality Assurance - A practical approach, 1994 (Prentice-Hall International, Hemel Hempstead).

30. Fenton, Norman E., Software Metrics - A rigorous approach, 1991 (Chapman & Hall, London).

31. Möller, Karl-Heinrich and Paulish, Daniel J., An Empirical Investigation of Software Fault Distribution, In Fenton, Norman, Whitty, Robin, and Iizuka, Yoshinori (eds.), Software Quality Assurance and Measurement A Worldwide Perspective, 1995, (International Thomson Computer Press, London), Chapter 21, p.242-253.

32. Shepperd, M., Products, processes and metrics, Information and Software Technology, 1992, Vol. 34, No. 10, p.674-680.

33. Miller, C., The Quantum study, In Ross, M., Brebbia, C.A., Staples, G. and Stapleton, J. (eds.), Proceedings of the 1st International Conference on Software Quality Management, 1993, p. 475-488.

34. Watts, R., Measuring Software Quality, 1987, NCC Publns.

35. Gillies, Alan C., Software Quality - Theory and management, 1992, (Chapman & Hall, London).

36. Fenton, Norman and Whitty, Robin, Introduction, In Fenton, Norman, Whitty, Robin, and Iizuka, Yoshinori (eds.); Software Quality Assurance and Measurement A Worldwide Perspective; 1995 (International Thomson Computer Press, London) Chapter 1, p.1-18.

Experience with Software Quality / Usability Metrics

Judy Lau and David N. Wilson

Judy Lau is a consultant with Shanable Pty. Ltd., 27 Duff Street,
Turramurra, NSW 2074, Australia.
Tel: +61-2-9402-8628 Fax: +61-2-9144-3630
e-mail: jslau@ozemail.com.au

David Wilson is a Senior Lecturer in the School of Computing Sciences,
University of Technology, Sydney, PO Box 123, Broadway, NSW 2007,
Australia.
Tel: +61-2-9514-1832 Fax: +61-2-9514-1807
e-mail: davidw@socs.uts.edu.au

Abstract

Several usability inspection methods and usability metrics to evaluate Graphic User Interface (GUI) systems have been developed. This paper describes an investigation to define an effective approach to usability evaluation – effective both in terms of being easy and cheap to implement and in terms of identifying usability improvements.

The approaches investigated were the heuristic evaluation method and the layout uniformity and essential efficiency metrics. These approaches were applied during the enhancement of an existing GUI system and were shown to work together effectively to identify usability problems and suggest usability enhancements.

1. Introduction

Graphic User Interface (GUI) systems for Personal Computers have been increasingly popular over text-based systems since the introduction of the Macintosh Desktop, followed by Microsoft Windows and IBM OS/2. Applications are now built with pictures, text, buttons and other graphical components. User operations can be performed with a mouse, pen, scanner, laser pointer, etc.; as well as a keyboard. All these features make the GUI systems more attractive and interesting to use. However, a system with attractive screens does not mean that it

C. Hawkins et al. (eds.), *Software Quality Management VI*

is usable. Too many components on a screen will make it complex, thus difficult to read. Too many steps to complete a function or task will be difficult to remember and use. Putting irrelevant information on the screen will distract and confuse the users.

Usability is defined as the measure of how effectively and efficiently a (software) product can be used by specified audiences with certain tools for certain tasks in particular environments [1]. A usable system has the following characteristics [2]:

- **Easy to learn and use** - The system is intuitive to use and easy to learn so that little or no training is required for new users.
- **Efficient to use** - Users can complete their tasks quickly and accurately without requiring to go through some awkward and obscure sequences.
- **Easy to remember** - The system should be easy to remember so that casual and occasional users can use the system without requiring learning it all over again even after having not use it for a while.
- **Flexible** - The system supports various task completion strategies for different types (experienced, normal, novice) of users and can be customised to different levels.
- **Minimum errors** - The system protects users from making costly errors and provides users opportunities and assistance to correct errors.
- **Pleasant to use** - Users enjoy using the system.

Usability is part of the total quality of a product (quality of use). It directly benefits the users and their organisations. Systems that are neither easy nor efficient to use will be costly to the organisation. This is because they can cause user frustration, errors and reduce their productivity and performance. They can also represent failed investments for the developer and impose undue stress on the work force [3]. Moreover, these systems will be difficult and expensive to support and maintain. Usability evaluation and testing can help to identify the usability defects of the system, improve the design and provide developers and designers a better understanding of how to design usable systems.

This paper describes an investigation to define an effective approach to usability evaluation – effective both in terms of being easy and cheap to implement and in terms of identifying usability improvements. The approaches investigated were the heuristic evaluation method and the layout uniformity and essential efficiency metrics. These approaches were applied during the enhancement of an existing GUI system and were shown to work together effectively to identify usability problems and suggest usability enhancements.

2. Usability Methodologies and Approaches

2.1 Usability Evaluation

There are four basic ways of evaluating user interfaces [4]:

- **Empirical methods** - The most commonly used methods where usability is assessed by testing the interface with real users. Empirical data is collected by observing the users' interactions with the system under evaluation. Very often, this is done with the use of video taping either at the users' environment or at a usability laboratory. Though empirical methods are commonly used, they are

costly and time consuming. This is because users can be difficult and expensive to recruit to perform testing on every aspect of all versions of a user interface.
- **Automatic methods** - Compute usability measures using evaluation software. Usability testing is a recent research area. There have been some experiments using evaluation software [5 - 7] but no evaluation software is currently commercially available.
- **Formal methods** - Calculate usability measures with the use of exact models and formula. Formal methods can be difficult to apply because they do not scale up well for complex, highly interactive user interfaces [4]. Usability metrics belong to this category.
- **Informal methods** - Evaluate usability based on rules of thumbs and the evaluators' skill, knowledge and experience. Usability inspection methods belong to this category.

This investigation focused on usability inspection methods (informal methods) which are more accessible to system developers, specifically the heuristic evaluation method, and usability metrics (formal methods) which can provide quantitative measures of usability, specifically the layout uniformity and essential efficiency metrics.

2.2 Usability Inspection Methods

Usability inspection is the generic term for a set of methods having evaluators inspect or examine the usability-related aspects of a user interface [4]. Usability inspections can be performed on a paper interface, a minimal prototype or full functioning prototype. The development of new inspection techniques has been growing rapidly since the first two methods: Heuristic Evaluation [8] and Cognitive Walkthroughs [9] were formally presented at ACM CHI'90 (Computer-Human Interaction) conference in Seattle. The main objective of these inspection methods is to identify the usability problems in an existing user interface design, then suggest solutions based on the inspection results to fix the problems and improve the usability of the design. Over the last few years, a number of usability inspection methods have been established : pluralistic walkthrough [10], consistency inspection [11], standard inspection [4], formal usability inspection [12], feature inspection [4], and guideline reviews [4].

Heuristic evaluation is a basic usability engineering method that is very easy and simple to use. It requires a small set of evaluators to examine the interface and judge its compliance with a list of established usability principles, the "heuristics" [13]. The original list of usability heuristics, shown in Table 1, was developed by Molich & Nielsen [8]. The last heuristic, help and documentation, was added to the list in 1991.

The evaluation process consists of four phases: a pre-evaluation training session, the actual evaluation, a debriefing session to discuss the outcome of the evaluation, and a severity rating session during which the evaluators assess the severity of the usability problems found. The severity of a usability problem is rated based on the following [13]:
- the frequency with which the problem occurs;
- the impact of the problem when it occurs; and
- the persistence of the problem.

1.	Simple and Natural Dialog
2.	Speak the users' language
3.	Minimise the users' memory load
4.	Consistency
5.	Feedback
6.	Clearly marked exits
7.	Shortcuts
8.	Precise and constructive error messages
9.	Prevent errors
10.	Help and documentation

Table 1: Original List of Usability Heuristics [13]

The list was revised in 1994 [14], derived from a factor analysis of 249 usability problems [13]. The revised heuristics and their relationship to the original heuristics are shown in Table 2.

Nielsen [2] recommends having three to five evaluators. This is because one person will never be able to find all the usability problems in an interface and using a large number of evaluators would not gain much additional information. Figure 1 shows a curve of proportion of usability problems found against the number of evaluators used based on the average of six case studies of heuristic evaluation. It also indicates why three to five evaluators are recommended.

Revised list of heuristics	**Corresponding heuristic in the original list**
1. Visibility of system status	Feedback
2. Match between system and the real world	Speak the users' language
3. User control and freedom	Clearly marked exits
4. Consistency and standards	Consistency
5. Error prevention	Prevent errors
6. Recognition rather than recall	Minimise the users' memory load
7. Flexibility and efficiency of use	Shortcuts
8. Aesthetic and minimalist design	Simple and natural dialogue
9. Help users recognise, diagnose and recover from errors	Precise and constructive error messages
10. Help and documentation	Help and documentation

Table 2: Revised List of Heuristics [13]

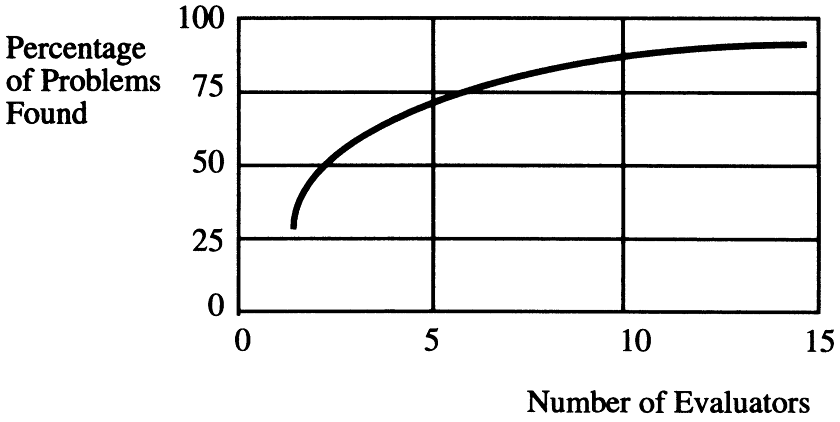


Figure 1: Usability problems versus number of evaluators [2]

Heuristic evaluation requires the individual evaluator to inspect the interface alone and record the problems and the severity ratings together with references to the violated usability principles. The evaluator goes through the interface several times but at least twice. The first pass is to have a feel for the flow and general scope of the system. The second pass is to focus on its individual dialogue element. The evaluators will meet to discuss and aggregate their findings only when all the interfaces are evaluated. This is to ensure the evaluations are independent and unbiased. The severity ratings collected will help to prioritise the list of usability problems to be fixed.

Heuristic evaluation does not provide a systematic way of fixing the problems or assessing the redesign of the system [2]. However the explanation given to each identified problem can usually provide insights to redesign and improve the system.

2.3 Usability Metrics

Usability metrics allow system designers and developers to evaluate the usability of user interface designs quantitatively. These metrics can be used to evaluate new designs and revisions; to compare alternative approaches or to assess the progress in successive refinements [15]. They can also help to make better user interface design decisions and solve usability problems [16].

The Essential Usability Metrics Suite is a collection of usability metrics developed from the Interactive Metric Project by Lockwood and Constantine [16]. The idea behind this research project is to develop a practical suite of metrics that are simple to use, conceptually sound, with a clear and transparent rationale connecting them to established principles of good design. The suite includes three task-sensitive metrics (essential efficiency, task concordance and task visibility), a structural metric (layout uniformity) and a content-sensitive metric (visual coherence) [15].

Essential efficiency is derived from essential use case modeling [17]. A well-designed user interface would require only as many steps as expressed in the essential use case. Essential efficiency is the ratio of the number of steps in the essential use case to those that are required to perform the task with a given user interface (see formula below). It measures how close a given design is to the essential use case model, in terms of the number of steps involved to perform a task or collection of tasks. It can be computed for a given task or as a weighted average based on a mix of tasks [15].

Essential Efficiency = Essential Length / Operational Length

where Essential Length represents the essential minimum number of steps from users' perspective.

Essential efficiency computed for a given user interface design with a value of 1 or close to 1 indicates that the design is efficient in performing the given task(s).

Layout uniformity is a structural metric that measures the uniformity or orderliness of the user interface layout. It is based on the rationale that usability is hindered by highly chaotic arrangements. Layout uniformity focuses on the spatial arrangement of the interface components. A user interface with the score of 0.6 to 0.9 is considered to be a good interface.

$$\textbf{Layout Uniformity} = 1 - \frac{(N_{size} + N_{top} + N_{left}) - A}{3 * C - A}$$

where C is the number of components on the screen

Nsize, Ntop and *Nleft* are the number of different sizes, top edge alignments and left edge alignments of components respectively

$A = \lceil 2\sqrt{C} \rceil + 1$ is an adjustment for the minimum number of possible alignments and sizes

3. The Study

3.1 Background Information

The sample system is a computerised resource planning, booking and scheduling system. The system is a GUI client/server system which runs in a Windows environment – the GUI front end was developed using Visual Basic. The development team derived a screen design guideline based on the Visual Design Guide that comes with Visual Basic and the Windows 3.1 interface standards. The system provides online help that links each screen to the corresponding help information and related topics. The members of the development team were on roster to support the calls from users during business hours. The project wàs initiated in 1993 : Phase 1 was implemented in August 1994; Phase 2 was implemented in February 1995; and Phase 3 was implemented in July 1995.

After Phase 3, a system performance review was conducted and as a result a usability evaluation was initiated. The objectives of the usability enhancements were to:

- minimise the number of screens and keystrokes used;
- provide feedback for long processes;
- provide new useful features;
- ensure the same look-and-feel throughout the system; and
- improve the usability and quality of the system cost effectively.

3.2 Usability Evaluation

The following steps were taken to identify the usability problems:

1. Inspect the system using heuristic evaluation to see if the system violated any of the usability principles.

2. Review and analyse the support logs and compile a list of problems reported, suggested solutions and users' feedback.

3. Collect usability problems found by other team members based on their individual knowledge and experience with the system.

4. Arrange meetings with the representative users to discuss any usability problems that they had found or were aware of.

The first three steps involved only the development team and were carried out before involving the users. The heuristic evaluation was performed with a focus more on tasks than on a single interface – the task scenarios that had been prepared for system testing were reused.

The evaluation process went smoothly. Usability problems were found in fifteen tasks using heuristic evaluation and six usability problems were identified from the support logs. Another evaluator found eight functions that required usability enhancements including three new functions that would be useful to the users. The support logs provided very useful information such as the most frequently reported errors and the steps taken to cause the errors.

The output from the heuristic evaluation was a list of usability problems, annotated with references to the usability principles that were violated (some violated more than 5 of the 10 principles). A preliminary usability problem report was prepared, listing the usability problems, the severity ratings (based on the frequency of the problem, the impact of the problem when it occurs, and the persistence of the problem) and the suggested solutions (based on the guidelines provided by the principles that were violated).

The users then presented their list of problems and suggestions without reference to the preliminary problem report. The users identified thirteen problems.

Comparison with the preliminary problem report showed that five problems identified by heuristic evaluation and support logs were also identified by the users. In total, twenty-eight problems were identified as shown in Tables 3a and 3b (new features are marked by '*').

Usability Problems		Step 1	Step 2	Step 3	Step 4
System Logon - too many steps and dialogs		x	x	x	
Open Another Job Sheet - too many steps and dialogs		x			x
Open Jobs for same Project - require to scroll down the list and may lose track of which job was previously opened.			x		x
Edit Clashes - change facility	*		x		x
Changing Confirmed Jobs and Bookings - warning messages prompted for each change					x
Schedule Information Display - Job name is not shown					x
Copy Job - too many steps		x			
Copy Bookings - too many steps		x			
Select Current/Future Jobs - cannot select multiple jobs		x			
Usage Log Load - cannot stop the process until the loading is finished		x			
Booking Number, Job Group - obsolete information		x	x	x	x
Select Jobs for Usage - should not allow users to select future jobs		x			
Cancel Job, Cancel Jobs - inconsistent		x			
Preview Jobs, Preview Potential Jobs - inconsistent		x			
Reports Identifier - System identifier and report number are not shown				x	x
Startup Message - to be maintainable by Federal Administrator	*			x	

Table 3a: Usability Problem Identified

Usability Problems		Step 1	Step 2	Step 3	Step 4
Move Bookings to Different Project and Job - can only be done by canceling the bookings and then re-entering the bookings to the desired project and job	*				x
Schedule Load - cannot stop the process until the loading is finished		x			
Provide Toolbar, Shortcut keys to speed up the interaction		x		x	x
List Box Selection - automatically highlight the entry if there is only one entry		x	x		
Major Facility Report - aligned the bookings by shift hours to improve readability	*				x
Project Booking Schedule - new report required	*				x
Administration System - New functions	*			x	
Include release notes for the last five releases in the Online help	*			x	
Review error messages to provide more meaningful information when errors occur		x	x	x	
Use different colour for different Job Status					x
Allow users to enter digits only for date fields and time fields					x
Cancel Booking / Cancel Bookings - inconsistent		x			

Table 3b: Usability Problem Identified

3.3 Usability Enhancements

The usability problems were classified as follows: task inefficiencies, system inconsistencies, failure to provide system feedback for long processes, and new features required. These classifications were then used to determine the usability evaluation method(s) that should apply for testing the revised designs:

1. For task inefficiency problems, apply essential use case modeling and retest the interfaces using the essential efficiency metric.

2. Redesign the interfaces (or design new interfaces for new features) from the usage point of view, using the usability heuristics as a guideline.

3. Test the layout structure using the layout uniformity metric to ensure the screen components are neatly arranged.

3.4 Example User Interface

The System Logon function (task inefficiency problem identified) is used as an example to illustrate Heuristic Evaluation, Essential Efficiency and Layout Uniformity for usability evaluation.

3.4.1 System Logon - Overview

System users have to be registered before they can use the system – depending on the their role, some users may have access to only one application and one database while power users may have access to multiple applications and one or more databases per application. All users must identify themselves by entering their 'username' and 'password' correctly in order to access the system.

With the system under evaluation, the users had to go through three screens before getting to the Main Menu:

1. Logon Screen: user enters 'username' and 'password' to provide his or her identity to logon to the system.

2. Application Selection Screen: user selects an application (most of the users will have only one selection).

3. Branch Database Selection Screen – user selects a database for the system (some users will have only one selection).

3.4.2 Usability Problems

The main problem with the logon process is inefficiency – users are required to go through too many steps and screens to access the system. Firstly, each selection screen requires a selection (highlighting the entry) and confirmation (clicking the OK button) for the user to proceed. For users who have only one application to launch and work on only one database, it is inefficient and unnecessary to make a single selection. Secondly, each screen must connect to the database in order to display the selection list and disconnect from the database when finished. Each database connection and disconnection takes at least a few seconds, depending on the network traffic. As a result, the database connection and disconnection has created an overhead for starting or quitting an application. Thirdly, power users who have access to multiple applications and multiple databases can see the list of available databases only after accepting an application selection. If they want to browse through the available databases for each application, or a wrong selection is made, they have to go back to the application selection screen and try another application. The logon process has violated heuristic principles 5, 6, 7 and 8 (see Section 2.2).

3.4.3 Essential Use Case and Essential Efficiency

Essential Efficiency is a task-sensitive usability metric which measures how close a given design is to the essential use case model, in terms of the number of steps involved to perform a task or collection of tasks [15]. It is the ratio of the number of steps in the essential use case to the number of steps that are required to perform the task in a given user interface. The computation is therefore very simple once the essential use case is defined by applying Essential Use Case Modeling. It is the perfect metric for evaluating interfaces in order to ensure task efficiency.

For the logon process, the number of steps used in the essential use case (Essential Length) is 3 and the number of steps required in the system interface (Operational Length) is 6 – as shown in Table 4. The Essential Efficiency for the logon process is, therefore, 0.5 (3 / 6). A good design should have an Essential Efficiency close to 1; 0.5 indicates that the interfaces use achieves only half the efficiency of the essential process.

3.4.4 Redesigning the Interface

With the problems identified and the essential use case defined, the next step was to redesign the interface in order to improve the usability. Two user roles (normal users and power users) were involved and the redesign had to take the following scenarios into consideration:

- User has access to only one application and one database.
- User has access to one application but multiple databases.
- User has access to multiple applications and one or more databases per application.

The interface was redesigned with the aim of using only as many steps as used by the essential use case and by applying the Heuristic Principles 3, 5, 6, 7 and 8. The following enhancements were recommended:

1. After users have logged on successfully, allow them to select an application and database on the same screen. This is to combine the application and database selection screens so that only one database connection and disconnection is required. This can also simplify the procedures involved and give users a clearer picture of which application and database they are selecting. Rather than confirming the selection one at a time, the users can confirm the selections in one step (Heuristic Principles 3, 7 and 8).

2. For users with permission to access only one application and one database, the application and the database will be highlighted automatically. This can save users two steps to select the only possible options (Heuristic Principle 7).

Essential Use Case		**System Interface**	
User Action	**System Response**	**User Action**	**System Response**
Request		Request	
	Show Logon dialog		Show Logon dialog
Identify self		Identify self	
	Verify identity		Verify identity
	Offer choices		Offer Application choices
Confirm selection			
	Start application	Select Application	
	Display Main Menu		Start application
			Offer Database choices
		Confirm selection	
			Display Main Menu
		Select Database	
		Confirm selection	
Essential Length	**= 3**	**Operation Length**	**= 6**
		Essential Efficiency	**= 0.5 (3 / 6)**

Table 4: Essential Efficiency

3. For users with permission to access more than one application, the available database(s) are displayed once they select an application. Once again, if there is only one database for the application, the database will be selected automatically. This provides users with the flexibility to browse through the available applications and databases without requiring to go back and forth between two screens. Also, the users are not be penalised for the time taken to connect and disconnect the database (Heuristic Principles 3, 5, 6 and 7).

4. The system provides the option to save the default application and database selections. This allows users to save their default application and database so that the system can select the saved selection next time the user logson to the system. This feature is especially useful to power users. They can simply confirm the default selection or change the selection if necessary (Heuristic Principles 3 and 7).

5. When both an application and a database are selected, the Start Application Button is enabled and set as the default button (that is, if user presses ENTER, the Start Application Button will be selected). This gives users the flexibility to press ENTER or click the Start Application Button to start the application. This also means that users can logon to the system using the keyboard without requiring to switch to a mouse (Heuristic Principle 7).

6. A startup screen with System Logo, Version Number, Copyright notice and a system message will then be displayed for a specific time interval. This is to provide feedback to the users showing which application they have started (Heuristic Principle 1). The Main Menu will be displayed when the application has started.

7. The system message and the time interval for displaying the startup screen can be maintained by System Administrator through the Administration System (Heuristic Principle 3).

In order to ensure the revised interfaces are usable, Layout Uniformity and Essential Efficiency were applied to check the layout structure and task efficiency respectively. The calculation of Layout Uniformity for the revised application and database selection screen is shown in Table 5. This revised interface has a score of 0.6 which indicates the screen components are neatly arranged.

Components	**Value**
Number of components (C)	9
Number of different sizes (Nsize)	5
Number of different top edge alignments of components (Ntop)	5
Number of different left edge alignments of components (Nleft)	5
Adjustment for the minimum number of possible alignments and sizes $A = \lceil 2\sqrt{C} \rceil + 1$	7
Layout Uniformity **1 - [(Nsize + Ntop + Nleft - A) / (3 * C -A)]**	**0.6**

Table 5: Layout Uniformity

The Essential Efficiency scores for different scenarios are shown in Table 6. The scores vary from 0.6 to 1, with a worst case score of 0.6. However, this is still better than the original interface where the score is 0.5; moreover, with the save default option, the efficiency can be improved to 1 if the users have saved their default selections.

Scenario	Operation Length	Essential Efficiency
1. Users can access only one application and one database	3	1
2. Users have access to one application and multiple databases (no default selection)	4	0.75
3. Users have access to multiple applications and one database for the selected application (no default selection)	4	0.75
4. Users have access to multiple applications and multiple databases for the selected application (no default selections)	5	0.6
5. Users have access to one or multiple applications and one or multiple databases for the selected application (default selections are saved)	3	1

Table 6: Essential Efficiency (for each scenario for the redesigned interface)

3.5 Enhancement Implementation

The twenty-eight enhancements were implemented in two phases. Phase 1 included redesigns to address the first sixteen problems in the list, while Phase 2 implemented the remainder. Before the implementation of Phase 1, the users were consulted – the redesigned system was demonstrated to local users and the usability enhancements document was sent to interstate users. In this document, the problems found and the corresponding recommended enhancements were described and the interfaces for both the existing and redesigned interfaces were included for easy references. The users agreed with and approved the enhancements.

Phase 1 was implemented in July 1996. Excellent user feedback was received. They were very pleased with the improvements which helped them to do their work with increased ease and efficiency : the number of steps to perform several tasks had been minimised; lengthy processes would display progress and allow users to stop the process if necessary; new and useful features were implemented; and inconsistency problems were fixed. As a result, the usability and quality of the system was improved. Users started to enjoy using the system. The number of support calls was reduced by 65% to 70% (calculated from the support calls received over a period of 56 weeks before and after the implementation). Most importantly, the productivity of both the users and the developers had increased by approximately 30%.

4 Conclusion

The fact that the usability evaluation methods (Heuristic Evaluation, Layout Uniformity and Essential Efficiency) were applied to an existing system provided an opportunity to study the effectiveness of these methods. The nearly two years of operational experience with the systems had provided: empirical data such as user feedback and problem logs; a fully interactive system for evaluation (the evaluation was not limited to paper prototypes); and users with a reasonable idea of how usable the existing system actually was. Taking the total of twenty-eight usability problems identified as the total number of problems, Table 7 illustrates the effectiveness of the different usability evaluation methods applied.

Heuristic Evaluation uncovered more than half of the total usability problems found (53.6%). However, the evaluation was performed on the system test task scenarios and did not identify any of the new functions – new functions would usually be found by domain experts and users, but not by any usability evaluation methods. Adjusting for the seven new functions identified in the total problems, Heuristic Evaluation identified 71.4% of the total problems found. Heuristic Evaluation is then an effective inspection method which is simple and inexpensive to use. Developers can inspect the design individually or in groups to uncover most of the usability errors before presenting the design to the users. Experience suggests that experienced evaluators are more effective than inexperienced ones; however, a small inspection team of three to five inexperienced people can uncover more usability problems than one experienced inspector. Whilst not yet a substitute for user testing, evaluations appear to be quite effective for detecting problems and generating insights into the usability of a developing interface [4].

Usability metrics allow system designers and developers to evaluate the usability of user interface designs quantitatively. The design metrics can be used together with usability evaluations where the evaluations identify the problems and the metrics evaluate and compare the redesigns.

Step	Method	Number of problems found	% of total problems found	Additional problems found
1	Heuristic Evaluation	15	53.6	N/A
2	Support Logs	6	21.4	2
3	Individual's knowledge and experience	8	24.6	4
4	User interview	13	46.4	7

Table 7: Summary of Usability Evaluation Results

These methods are inexpensive to use and can produce very useful results. In addition, these methods have the following characteristics [15]:

- easy and simple to use;
- applicable to paper prototypes and design models;
- direct guidance for design;
- effective prediction of actual usability in practice;
- direct indication of relative quality.

Usability metrics allow system designers and developers to evaluate the usability of user interface designs quantitatively. The design metrics can be used together with usability evaluations where the evaluations identify the problems and the metrics evaluate and compare the redesigns.

In summary, Heuristic Evaluation, Layout Uniformity and Essential Efficiency are considered as a set of cost effective usability evaluation methods which can be used when enhancing existing interfaces or designing new systems. Without the right design, systems can be difficult to use. Enhancing the usability of a system can improve the productivity and reduce the support and maintenance costs. Developers should not simply focus on developing functional systems but usable systems as well. In order to improve the usability of interactive systems, design must start with the end users and usability engineering should be included in the software development life cycle.

References

1. Lingaard. Usability Testing and System Evaluation. Chapman & Hall, London, 1994.

2. Nielsen, J. Usability Engineering. Academic Press, Inc., 1993.

3. Melchior, Bosser, Meder, Koch & Schnitzler. Usability Study: Handbook for practical usability engineering in IE projects. ECSC-EC-EAEC, Luxembourg, 1996.

4. Nielsen, J. & Mack. Executive Summary. In: Nielsen, J. (Ed). Usability Inspection Methods. John Wiley & Sons, Inc., 1994.

5. Comber & Maltby. Screen Complexity and User Design Preference in Windows Application. In: Proceedings of OZCHI'94, Canberra, 1994.

6. Comber & Maltby. Evaluating Usability of Screen Designs with Layout Complexity. In: Proceedings of OZCHI'95, Canberra, 1995.

7. Sears, A. Layout Appropriateness: A Metric for Evaluating User Interface Widget Layout. IEEE Transactions on Software Engineering, July 1993; **19** (7): 707-710.

8. Molich, R. and Nielen, J. Improving a human-computer dialogue. Communications of the ACM, March 1990; **33** (3): 333-348.

9. Lewis, Polson, Wharton & Rieman. Testing a walkthrough methodology for theory-based design of walk-up-and-use interfaces. In: Proceedings of ACM CHI'90 Conference, Seattle, WA, 1-5 April 1990: 235-242.

10. Bias, R. The Pluralistic Usability Walkthrough. In: Nielsen, J. (Ed). Usability Inspection Methods. John Wiley & Sons, Inc., 1994.

11. Wixon, Jones, Tse & Casaday. Inspections and Design Reviews: Framework, History, and Reflection. In: Nielsen, J. (Ed). Usability Inspection Methods. John Wiley & Sons, Inc., 1994.

12. Kahn & Prail. Formal Usability Inspections. In: Nielsen, J. (Ed). Usability Inspection Methods. John Wiley & Sons, Inc., 1994.

13. Nielsen, J. Heuristic Evaluation. In: Nielsen, J. (Ed). Usability Inspection Methods. John Wiley & Sons, Inc., 1994.

14. Nielsen, J. Enhancing the explanatory power of usability heuristics. In: Proceedings of ACM CHI'94 Conference, Boston, MA, 24-28 April 1994.

15. Constantine, L. Measuring the Quality of User Interface Designs. In: Proceedings of Software Development '96, 28 March 1996.

16. Constantine, L. Usage-Centered Software Engineering: New Models, Methods and Metrics. In: Purvis, M. (Ed.). Software Engineering: Education & Practice. IEEE Computer Society Press, Los Almitos, CA, 1996.

17. Constantine, L. Essential Modeling - Use Cases for User Interfaces. Interactions, April 1995.

Improving Information Systems Development by Managing Project Risk

Peter Smart
Price Waterhouse, Information Systems Risk Management,
London, UK. E-mail: Peter_Smart@europe.notes.pw.com

Abstract

The effective management of business projects is increasingly seen as being a key aspect of strategic planning and implementation, especially where major integrated business change and Information Systems (IS) development are involved. The ability to deliver expected benefits will improve business performance and ultimately enhance shareholder value. A variety of project management approaches and assessment techniques have been employed in recent years which, directly or indirectly, contribute to the management of risks on such projects. However, used in isolation, these may not prevent the adverse impact of risks being felt on a project, programme or organisation as there may be no formal structure to assess and prioritise the potential impact, determine mitigating actions and implement process improvements. A Project Management framework is described which has proved to offer considerable benefit. The framework relies on the periodic assessment of risks and an appropriate organisational structure for supporting the framework. A Project Office provides centralised functions for effective implementation of the framework.

1 Introduction

The effective management of business projects is increasingly seen as being a key aspect of strategic planning and implementation, especially where major integrated business change and Information Systems (IS) development are involved. The ability to deliver expected benefits will improve business performance and ultimately enhance shareholder value.

For every project undertaken, there are associated business and project risks and often the failure to adequately manage these risks can have serious implications. Organisations can and do suffer major tangible (e.g.

C. Hawkins et al. (eds.), *Software Quality Management VI*

financial) or non-tangible losses (e.g. reputation). Senior management need assurance that all projects, especially large system implementations, are well controlled and that the associated risks are minimised.

When organisations undertake larger or more strategically critical programmes the associated greater size, complexity and risk must be effectively managed to ensure reaping the higher business rewards. The challenges associated with such programmes may include:

- a number of interdependent IS, business and infrastructure projects running in parallel;
- greater management structure and division of management responsibilities;
- integration and cohesion of management processes across the programme;
- information volume and timeliness: decision making, escalation, reporting;
- greater scope and complexity of business changes and benefits.

During recent years the approach to addressing these challenges on large projects has been to formalise the development process using project management methodologies, software development lifecycle models, and certification or assessment schemes. Often these will be applied in combination and, whilst there may be a provision for improving the quality of the processes employed, the benefits of doing so may not be readily appreciated on a particular project or there may be an acceptance that there is not time to do so.

This paper introduces the concept of a project management framework and goes on to describe how this may be applied to address and control the risks introduced by the challenges described above.

The project management framework provides a generic model of IS development. The framework is characterised by a number of capabilities, some of which may be met by existing methodologies and techniques within an organisation, while others may need to be implemented. A key element of this approach is the identification and prioritisation of risks, particularly those affecting the management processes. By first addressing those risks which may potentially have the greatest impact on an organisation - by establishing appropriate mitigation actions, capabilities and controls - the framework is established to progressively improve the management processes and quality of deliverables.

In order to maximise the benefits of implementing the Project Management framework, experience has shown that the organisational structure to support it may need to be reviewed. In order to maximise the synergy within a project and across an organisation the support for the management processes are best provided centrally. The establishment of a Project Office, or the use of an existing similar entity provides a solution to this requirement.

2 Project Management Risks

High profile failures of major business system and IS development projects highlight the need for effective business project management. Publicised examples are the London Ambulance Service's command and control system, the London Stock Exchange's Taurus project and the UK National Insurance System. These have reinforced the belief, supported by many industry surveys, that most projects do not satisfactorily deliver the intended benefits within the planned timescale and cost. Figures quoted in [1] show that only 1.77% of software was used as delivered and 47.27% of software delivered was not successfully used. The remainder was either not delivered (28.8%), extensively reworked (19.2%) or used after changes (2.95%). With business programmes often involving major change and investments of strategic importance, it is critical that an effective project management framework is in place.

Many projects under-perform due to inadequate management processes, which may arise from weaknesses, omissions or poor implementation in the management controls, procedures and methodologies. Typical examples are:

- inadequate management of risks,
- inadequate overall project management structure with appropriate business and project accountability and responsibilities, and
- failure of project management to understand and address interdependencies between the various parts of the business and project processes.

This paper describes an approach that has been used to establish a project management framework and supporting infrastructure which enables an organisation to proactively manage project risks and improve the management processes.

3 Managing Risk and Improving Quality

Formalising a development methodology through the use of a documented quality management system is employed by many organisations to control the risks inherent in large-scale IS development [2]. By promoting a consistent approach throughout an organisation, verifying compliance and making changes where necessary, risk will be reduced and the processes will gradually improve. The ISO9001 standard, by means of the TickIT scheme [3] for IS projects, defines a model for a developing IS systems. Even without the constraints of a certification process the benefits of quality planning have been described [4]. Process improvement is supported through the use of these approaches although it is not directly facilitated.

The Capability Maturity Model (CMM) [5], SPICE [6] and similar schemes encourage process improvement by assessing an organisations capabilities against criteria which allows the level of maturity to be established and provides guidance for moving to the next higher level.

The improvement cycle may be too long to impact the current project or the organisational structure may be such that the synergy between projects can not be realised in a timely manner. Also many organisations who may formalise their methodology are not required to submit themselves for certification or assessment against these models and hence one of the key benefits of following this route, namely the independent identification of risks and improvement of the management processes, may not be realised.

In order to maximise the benefits of using these models the concept of a Project Management Framework has been defined.

4 Project Management Framework

4.1 Objectives

The key objectives of a project management framework are to ensure that:

- benefits and functionality expected by the business sponsors are delivered by the project;
- project risks are properly managed; and
- a comprehensive set of project controls is in place.

These objectives will only be met if several major project management critical success factors are addressed:

- delivery of the benefits with the requisite level of quality on schedule and within budget;
- maintaining tight control over projects through comprehensive planning driven by clear business case requirements;
- identification and communication of major issues and risks with sufficient lead time to react;
- co-ordination of cross-project and cross-function activities to minimise gaps and conflicts;
- support and buy-in of senior management;
- effective business communications, both internal to a project and between the project and its stakeholders.

4.2 Structure

There are a number of project management methodologies and techniques employed in projects and programmes, but regardless of how these are structured, project tasks can generally be grouped into three distinct process categories:

- Management,
- Delivery, and
- Support.

An effective project management framework should support and enhance the key **Management processes**. The **Delivery processes** relate to the actual creation of a product or service. The **Support processes** are those necessary to provide the project infrastructure in which the first two categories are conducted, for example configuration management, training and tool management. Communications occur between processes and consist primarily of statements of requirements, status reports and other project deliverables.

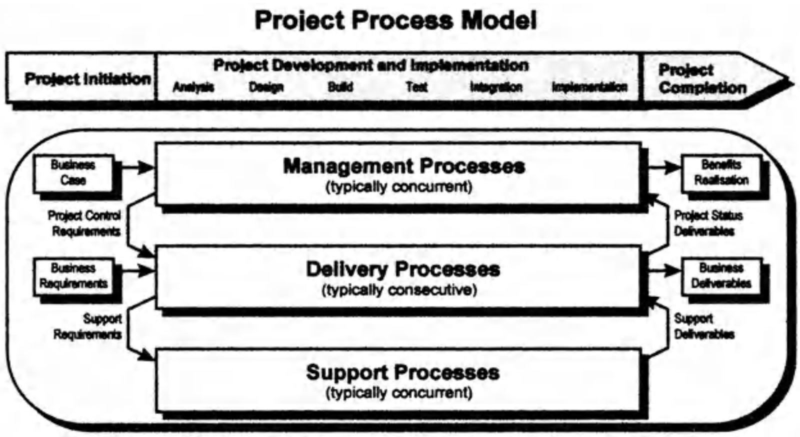


Fig 1. Project Process Model

4.2.1 Management Processes

The Management processes comprise a number of project management activities, typically performed concurrently throughout the duration of the project. An organisation wishing to observe good practice will ensure that the management processes encompass the following:

Project Management and Administration	Administration and document management,
	Project definition
	Contract and supplier management
	Project reporting
	Communications
Planning and Scheduling	Planning and estimating
	Status monitoring
	Budgeting
	Resourcing
Risk Management	Risk identification and assessment
	Risk mitigation
Quality Assurance	Quality management
	Acceptance management
Issue & Change Management	Tracking and escalation
	Impact assessment
	Change approval

4.2.2 Support Processes

The Support processes comprise a number of activities, typically performed concurrently throughout the duration of the project. Good practice covers:

Configuration Management	Management of project deliverables
	Version control
	Secure environment
Problem Management	Problem capture and impact assessment
	Progress chasing
	Integration with issue and change management
Operations Infrastructure	Construction of development, test and live operating environments
	Implementation
Toolset Management	Maintenance and management of support tools
Information Gathering	Timesheet tracking
	Project finances
	Tracking deliverables against plans
Human Resource Management	Staff and skills mix
	Administration of contractors
	Resource profile

4.3 Assessing and Reporting Risks

The project management processes undertaken by an organisation need to be assessed to measure their effectiveness and comprehensiveness. This can take the form of a brief healthcheck or a full review, with the objective of identifying risks and highlighting areas for improvement.

A project management review is used to assess project management processes, organisation and operation. To maximise the benefits, these reviews should be conducted at the start of the project and on an ongoing basis throughout the project lifecycle. The risks identified during the review may be used as a basis for implementing or enhancing a project management framework.

The key objectives of reporting on project management risks are to highlight to project managers the risks impacting their projects and to provide senior management with an indication of the effectiveness of the project management framework. Risks must be prioritised, presented clearly to management and mitigation actions and responsibility must be agreed to maximise the benefit.

4.4 The Improvement Cycle

The Project Management Improvement cycle provides an approach for establishing or improving an organisation's project management processes. It is unusual for an organisation to address all the actions required to mitigate all identified risks at one time - either because insufficient resources are available or the level of change would be too great. In practice, an iterative route may be chosen to expedite improvements and minimise disruption tot he organisation.

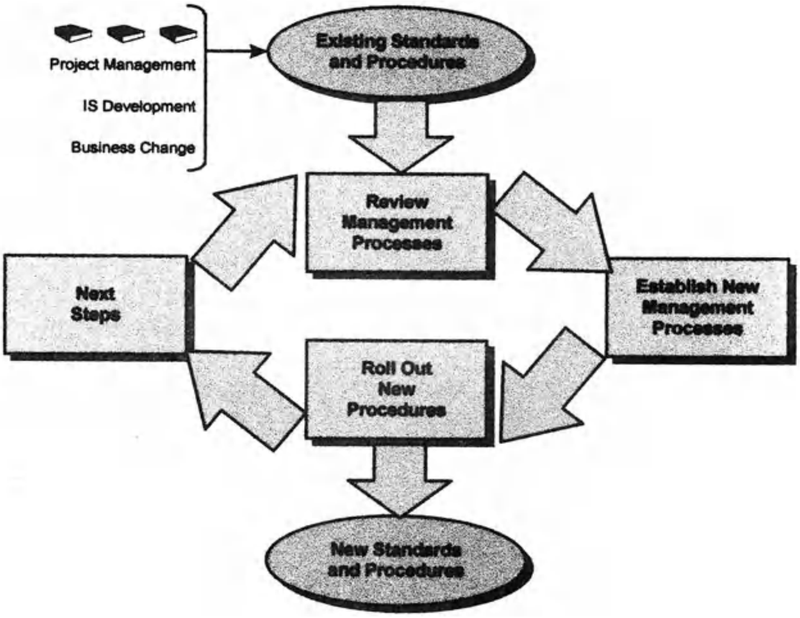

Project Management
IS Development
Business Change
Existing Standards and Procedures
Review Management Processes
Establish New Management Processes
Roll Out New Procedures
Next Steps
New Standards and Procedures

Fig 2. Project Management Improvement Cycle

The key stages in the improvement cycle are as follows:

- **Review management processes.** The improvement cycle starts with a review of existing Management processes using our proven Project Management Audit approach.

- **Establish new processes.** Working with pilot projects in the organisation, areas for improvement are identified, prioritised and agreed. New working practices are implemented and proved in a pilot environment and any fine tuning applied.

- **Roll out new procedures.** Having gained sufficient acceptance the practices are embodied into the framework standards and procedures and then rolled out to other areas of the business using appropriate communications and training forums.

- **Next steps.** Having addressed the highest priority needs, the review can be extended to other areas as required

A timetable for operating the cycle must be agreed and followed by management. The development of a project management framework should be viewed as an iterative approach which can be targeted and tailored towards the most urgent needs. A cyclic improvement process also permits an evolutionary development of the framework which facilitates less frantic management of change and aids acceptance by project managers and teams.

4.5 Implementation

Experience has shown that centralising the support for management processes within a Project Office, or similar organisational entity, provides an optimal approach to delivering these capabilities.

The Project Office sits between the project teams and the senior management team, and will typically provide project management support services to a large programme or to a number of related projects. It will have a range of responsibilities, including monitoring and reporting of status, tracking of risks, issues and changes, and the development and implementation of project standards.

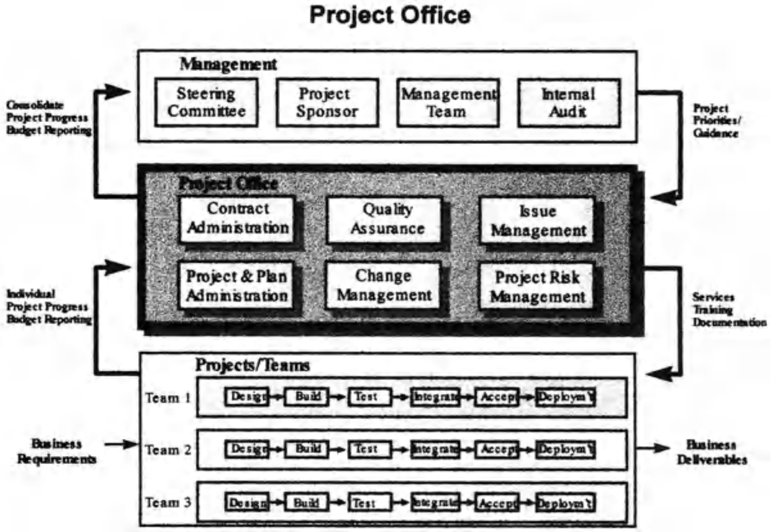


Fig 3. Project Office Organisational Relationships

4 Conclusions

Experience has shown that by establishing a project management framework, focusing on the risks which arise from Information Systems development offers a pragmatic solution to improving the software development process. People to introduce new processes, or to improve existing processes, are always scarce on any project as effort is focused on delivery. However, by identifying and then analysing risks, followed a managed cycle will allow an improvement action plan to be established. As projects progress, with new risks possibly materialising, regular reassessment of risks is necessary to ensure the resources available for improvement continue to be applied in the most effective area.

By targeting the challenges through the use of a project management framework to address the critical success factors for the project, the following benefits can be realised:

- better delivery of integrated business and IS changes and benefits;
- consistent and co-ordinated approach to best practice project execution and managing complexity;
- synergy across projects especially managing risks, prioritisation and allocating scarce resources;

- centre of excellence for the application of project management techniques and tools;
- communication of requirements for evolving management methods to IS and business project managers and to senior management.

Acknowledgements

The author would like to take this opportunity to thank his colleagues in the Information Systems Risk Management practice of Price Waterhouse for their assistance in the preparation of this paper. In particular thanks to Jeff Thompson and James Chrispin for their guidance and permission to publish this work.

References

1. Charette, Robert N. Software Engineering Risk Analysis and Management. McGraw-Hill, New York, 1989.

2. Smart, Peter J. Implementing a Quality Management System: A Case Study. Proceedings of the First International Conference on Software Quality Management, Southampton, 1993.

3. TickIT: A Guide to Software Quality Management System Construction and Certification Using EN29001. Department of Trade Industry, UK, 1992.

4. Ould, Martin A. Strategies for Software Engineering: The Management of Risk and Quality. Wiley, Chichester, 1990.

5. Paulk, M.C., Curtis, B., Chrissis, M.B. and Weber, C.V. Capability Maturity Model for Software Version 1.1. Technical Report CMU/SEI-93-TR-24, Software Engineering Institute, Pittsburgh, Pennsylvania, USA, 1993.

6. Dorling, A. SPICE: Software Process Improvement and Capability dEtermination. Software Quality Journal 1993; Volume 2 Number 4; 209-224.

Information Quality Supported by Meta Model

Tatjana Welzer, Ivan Rozman
Faculty of Electrical Engineering and Computer Science, University of Maribor
Smetanova 17, 2000 Maribor, Slovenia
e-mail: welzer@uni-mb.si, i.rozman@uni-mb.si

Abstract

In the last years the information technology has been spectacularly successful in automating many operations and making data available to more people. For this reason we are today faced with an increasing demand for more and more complex databases applications. The rapid growth has stimulated the need for a high level concepts, tools and techniques for a database design, development and retrieval with a final goal: better information quality.

One of the new possibilities is using a meta data model repository which enables more expedient and efficient design of the good quality data models. As a consequence, data quality as well as information quality of any information system improve.

1 Introduction

Over the years many of approaches to improve information quality have been developed and employed in various situations. It is important to recognize that most of these approaches are only vaguely aware that the prerequistive for the information quality is data quality, despite the fact that in the information technology aggressive steps to improve just the data quality are being taken.

In the last years the information technology has been spectacularly successful in automating many operations and making data available to more people. The advances of information technology also have the impact on poor data quality [1]. But unfortunately just as it is natural to assume that computerized data are correct, so too it is natural to blame the information technology when data are incorrect. These problems can grow especially in the data warehouses environments as well as on the Internet.

Further, data (information) users usually use data in their applications without giving the conceptual view much thought. But in opposite, a high quality conceptual view is of great importance for avoiding before mentioned problems and improving the data as well as the information quality.

C. Hawkins et al. (eds.), *Software Quality Management VI*

Conceptual or data modeling as a process of producing the conceptual view is a complex and a hard job that results in a data model (conceptual model) of an enterprise. In principle, each of the models designed by one of numerous approaches is used only once, i.e. within a single project. This seems to be very time consuming and not very practical, because a model or parts of a model derived for one enterprise could be used also in future projects [2]. This argument results in the meta data model repository which enables the use of models or submodels of previous projects in the actual design. Nowadays such models are quite often called patterns. Martin Fowler [3] defines a pattern, as an idea (the term idea is used to stress the fact that a pattern can be anything) which has been useful in one practical context and will probably be useful in others. Similar philosophy is behind the already mentioned meta data model named MetaBase repository [4], considering reusable database models.

The MetaBase project was stimulated by the assumption that the enterprises are similar or at least that they have similar components. By considering these similarities a data model of a high quality (the model was already used and for that reason also already tested in at least one project) could be used as a starting point and can then be adjusted for the original enterprise. This approach enables more expedient and efficient design of the good quality data models, consequence in a good data quality as well as information quality of any information system.

A brief overview of reuse on which the meta data model is based, is presented in chapter 2; the MetaBase concept is the main goal of the third chapter, while chapter 4 gives a concept of the search for similarities considering quality dimensions of reusable components. We finally conclude with a summary of the proposed concepts.

2 Reuse

In the field of reuse there is no single accepted definition for the most of the terms connected with this field. Therefore we have accepted for terms "software reuse" and "reusable software components" definitions from C. Brown [5]:

Software reuse means the use of an existing software component in a new context, either elsewhere in the same system or in an another system.

Reusable software component is a software entity intended for reuse; may be design, code or other product of the software development process. Reusable software components are sometimes called "software assets".

Subsequently, according to the above definitions we will try to give detailed explanation about terms "database reuse" and "reusable database" components [2]:

Database reuse is a process of using existing database conceptual models or parts of them rather than building them from scratch. Typically, reuse involves abstraction, selection, specialisation and integration of reusable parts, although different techniques (when defined) may emphasizes or de-emphasizes some of them.

The primary motivation to reuse database components (conceptual models or parts of them) is to reduce the time and effort required when building a conceptual model, actually a database. So as the quality of software systems is enhanced by reusing quality software artifacts (which also reduces the time and effort required to maintain software) similarly reusable database components influence further database design and (not at last) also the database maintenance. Enhancement of the database quality is expected.

Further, considering above definitions we should try to answer to the following question: Is the software reuse easy and the database reuse difficult? Actually, every reuse in the field of computer science is difficult. Useful abstractions for large complex reusable software components or reusable database models will typically be complex [2]. For this reasons an appropriate way to present reusable concepts, either software, artifacts or conceptual models or parts of them must be found.

3 MetaBase

Our approach to apply reuse concepts also to the database design is based on an object oriented meta data model. We have decided for object oriented paradigm in order to take advantages of this concept (reuse, inheritance, methods) to represent our meta data model.

The MetaBase *(Figure 1)* model is introduced as a three level model, which distinguishes object level, enterprise level and function level. The enterprise level is central in the MetaBase model. It contains the conceptual models and external models that describe the particular enterprise. Upon this central block of the MetaBase a function level is built into which business and functional domains are integrated. An application domain links the function and the enterprise level. On the other hand the enterprise level is also related with a subordinated object level across objects. The object level contains the representation of semantic objects, which constitute the conceptual and external models.

3.1 Enterprise level

It presents a connection to the functional level across the application domain and the enterprise. The application domain is a very important part of our structure because reuse of conceptual models (database components) is more promising if their application domains are the same or similar [6] while a domain model could be a terminological framework for discussing commonalties and differences between conceptual models of an enterprise within the application domain, considering the above presented outputs.

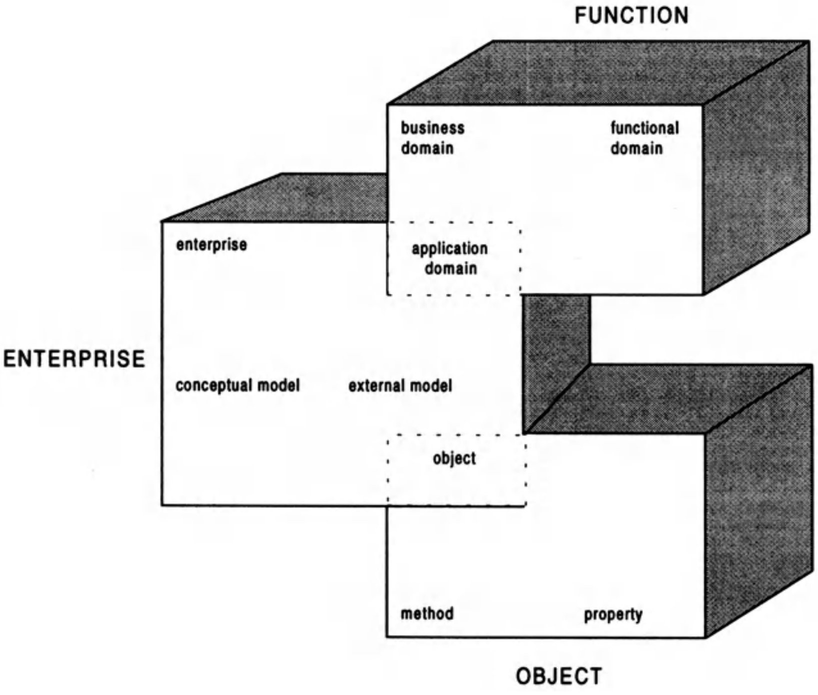


Figure 1: Concept of the MetaBase

External models are used for a number of varying purposes [7]. They represent the view of an application systems and user groups to the database, i.e. to the conceptual model. Like conceptual models they include a set of related objects. An external model is a submodel of one or more conceptual models. So we do not look on the derivation of an external model from a conceptual one, but rather on the result of the derivation. Therefore, an external model can be derived (by different transformations) from different conceptual models. Conceptual models as well as external models are connected with a respective application domain as well as the enterprise.

3.2 Functional level

Each application domain is characterised by classification in two dimensions: the business domain and the functional domain. In our meta data model both business domain and functional domains are organised in a generalisation hierarchy, the functional domain, furthermore, in an aggregation (part-of) hierarchy.

3.3 Object level

The structure we are discussing here *(Figure 1)* is designed according to the object oriented data model. The key reason for this decision was to include operations (methods) to make the structure more active and to open possibilities to integrate reuse of software and reuse of database design [8].

The meta data model reflects the associations usually found in object oriented data models, i.e. generalisation and aggregation. The "is-a" relationship between objects represents the generalisation (inheritance) lattice of an object oriented data model. The aggregation hierarchy is modeled through properties relationship, i.e. the components of a composite object are its properties. In our structure an object is characterized by several properties. Some of them are values but some of the properties are objects themselves. Further, the object behavior is presented by associating methods to objects. The introduction of methods enables us to go beyond solely static structures.

4 Search for similarities

The objectives of constructing the MetaBase structure is to assist database administrators during database design and software engineers during software design. In a reuse centered approach the designer has to search for components suitable to assist in the solution of the problem at hand in the components repository. The MetaBase assists the designer as it guides the search and provides powerful search and browsing capabilities. Thus it narrows the components space the designer has to search through. In our approach the final decision which components are suitable has always to be made by designer. The system helps to identify candidate components. In the following, we sketch the search process and the most important tuning parameters.

At first the designer narrows the search space by choosing business domain and functional domain and by identifying an application domain. All other search processes will be conducted within this application domain. Already here the designer has the possibility to narrow the search space by choosing an application domain from the bottom of the is-a hierarchy or broaden it by selecting more generic application domains through generic business and functional domains.

Within the set of objects of an application domain, the user can choose among alternatives [4]:

The user can browse through all the objects.

In particular, if this set is too large, he can browse through generic objects. Typically, the number of objects decreases from the lower levels of the hierarchy to upper levels. So the user can climb up the hierarchy to a level which is suitable for manual browsing. In this level the user can select and deselect generic objects, which are considered relevant for the problem. Climbing down the hierarchy, the system presents the subordinate objects of the selected generic objects to the user. The user selects appropriate objects at each level until the leaf objects are reached.

Starting from a single object (or a group of objects) the system can determine similar objects. The user directs this search by specifying the length of the path to a common ancestor to be considered.

The system can search for objects with similar properties as a given object.

The user can choose a set of conceptual models or external models and restrict the search to objects contained in the selected models.

The user can browse through the whole MetaBase: searching for all the properties of an object; which objects are contained in an external or conceptual model: in which models an object is contained, in which objects a certain property is referenced, and so on.

According to described sketch of the search strategies, the techniques for searching in the MetaBase is more formally introduced in [2].

4.1 Quality Dimensions of the MetaBase

It is obvious already from the presentation of the MetaBase repository (meta data model) that components saved in the repository are of at least basic quality. The intention is to reach an "ideal" conceptual model. For this purpose 15 characteristics should be fulfilled: relevance, obtainability, clarity of definition, essentialness, attribute granularity, domain precision, naturalness, occurrence identifiability, homogeneity, minimum redundancy, semantic consistency, structural consistency, robustness, flexibility [1] and according to the user needs some characteristics of his or her own could be added.

The before mentioned basic quality of the MetaBase components fulfill according to the list the following characteristics:

Relevance - objects needed by the applications are included in conceptual models.

Clarity of definition - all the terms used in the conceptual model are clearly defined.

Comprehensiveness - each needed attribute should be included.

Occurrence identifiability - identification of the individual objects is made easy.

Homogeneity, structural consistency - object level enables the minimisation of unnecessary attributes.

Minimum redundancy

Semantic consistency - models are clear and organised according to the application domains.

Robustness, flexibility - through reuse both characteristics are fulfilled.

Conclusion

Conceptual models of different databases are neither right or wrong. According to the pattern technology they are more or less useful. The useful part is called pattern and presents a starting point for a new design [3]. Similar reusable database components from the MetaBase repository present a starting point for conceptual database design supported by database reuse and of course according to the MetaBase view they are more or less reusable.

Even just as important as reusability or may be even more is the quality dimension of reusable components. According to the 15 quality characteristics the basic quality of the MetaBase components fulfill already 10. But the final goal on the way to the information quality are not the 15 characteristics, but Deming's fourteen points focused on data and information [1]. In further research on the meta data models repository, they should be considered in a way to assure better quality of conceptual models and reusable components.

References

[1] T.C. Redman: Data Quality for the Information Age, Artech House, 1996.

[2] T. Welzer: Model of the database system with explicit built in expert knowledge, Doctor of Science Thesis, University of Maribor, 1995.

[3] M. Fowler: Analysis Patterns: Reusable Object Models, Addison Wesley, 1997.

[4] T. Welzer, J. Eder: Meta Data Model for Database Design, In V. Marik, J. Lazansky, R. Wagner (Ed.) Database and Expert Systems Applications, Proceedings 4th International Conference, DEXA'93, Prague Czech Republic, 1993, pp.677-680.

[5] C.L.Brown, Reuse: In J.J. Marciniak (Ed.) Encyclopedia of Software Engineering, John Wiley & Sons, 1994, pp. 1055-1069.

[6] P. Freeman: Reusable Software Engineering Concepts and Research Directions, In P. Freeman (Ed.) IEEE Tutorial Software Reusability, 1987, pp. 10-25.

[7] M.Dobrovnik, J.Eder: A concept of Type Derivation for Object-Oriented Databases, In L.Gün et al (Eds.) Proceedings of the eight International Symposium on Computer and Information Science (ISCIS VIII), Istanbul 1993, pp. 132-148

[8] J. Eder, W.Rossak: An Archive of Semantic Data Models as Basis for Software Re-use, In Proc. 8th International Symposium on Applied Informatics, Innsbruck, Austria, February 1990, pp. 284-287.

Quality and RAD: A Contradiction in Terms?

by Paul Allen,
VP Methods

SELECT Software Tools Plc
Cheltenham, England

April 1998

Rapid Application Development (RAD) is commonly applied to delivery of point software solutions in isolation from enterprise-wide requirements. This is problematic in the context of large organisations seeking practical business-oriented software re-use, especially in a way which capitalises on legacy software. This article examines the role of architecture, and describes a component-based approach that facilitates re-use of services. The need for high quality and generality of components that are trusted by multiple consumers is increased by an order of magnitude and is set in vivid contrast to the need for fast solutions to pressing business needs.

The article highlights three particularly critical factors that have emerged in my practical project work: a clear and up to date software process based on the needs of component builders and solution assemblers; a service-based architecture for component-based development; an approach to assess risks against quality features in order to determine the level of rigor appropriate for a particular development relative to the perceived value of the software to be produced.

1. Introduction

1.1 Good News

RAD was originally coined as a response to the well documented problems of top-heavy waterfall based methods. As conceived by James Martin [1] and others RAD bought with it many useful techniques such as Joint Application Development (JAD), prototyping and timeboxing. Other good features of RAD include its removal of documentation overload and a "re-use before you buy/buy before you re-use" philosophy.

More recently, I have contributed to the work of the Dynamic Systems Development Method consortium (DSDM), which has forged good practical guidance on the responsible use of RAD stressing nine principles all of which are important if a quality system is to be supplied in the timescales required by the business [2]. The nine principles of DSDM are as follows:

C. Hawkins et al. (eds.), *Software Quality Management VI*

1. Active user involvement is imperative.
2. DSDM teams must be empowered to make decisions.
3. The focus is on frequent delivery of products.
4. Fitness for business purpose is the essential criterion for acceptance of deliverables.
5. Iterative and incremental development is necessary to converge on an accurate business solution.
6. All changes during development are reversible.
7. Requirements are baselined at a high level.
8. Testing is integrated throughout the lifecycle.
9. A collaborative and co-operative approach between all stakeholders is essential.

"All of these principles have been found to be necessary, if a quality system is to be supplied in the timescales required by the business" [3].

1.2 Bad News

However, in my visits to many organisations I find quality is all too often sacrificed in the rush to meet unrealistic deadlines under the banner of "RAD". This is the irresponsible side of RAD: "no modelling" or uncontrolled RAD based on hacking away in a GUI builder to produce fast results. Unfortunately those results tend to be unpredictable and difficult to maintain. Such an approach is short-term and in large enterprises can be dangerous, ultimately resulting in inconsistent interfaces, redundancy of information and a growing maintenance backlog.

1.3 The Attraction of Components

Component-based development, in which software solutions can be assembled in analogous fashion to hardware, is increasingly presented as the answer to these problems. However "there's no free lunch": the increasing complexity and scale of today's mainstream software development projects make pure RAD a highly risky option for component development. Coupled with the power of the component marketplace, uncontrolled RAD could in fact get very ugly, leading to a kind of software anarchy.

2 The Software Process

2.1 The Process Spectrum

Let's go back to basics for a moment. If you're approached in the street and asked to fill in a marketing questionnaire you may refuse or else fill it in as quickly as possible. On the other hand if you're completing your tax return you'll probably take great care and then copy it for reference next year so you can re-use much of the hard toiled information. In everyday situations the means must justify the end.

Software development is no different. All software processes can be considered to be somewhere on a spectrum of rigor as shown in figure 1. An ultra-RAD approach will result in software which will have so many bugs as to render it unusable. The long term consequences of ultra-RAD approaches are in fact self-defeating. Because solutions are always geared to immediate needs software never gets reused. This means that in the long term applications become increasingly inconsistent. The same code has to be replicated in an increasing number of places. Changes to that code are an increasing maintenance drain on valuable resources. Conversely, the ultra conservative approach will never actually deliver anything as management "pulls the plug" in frustration at development inertia or "analysis paralysis". Attempts at software re-use will be "theoretical".

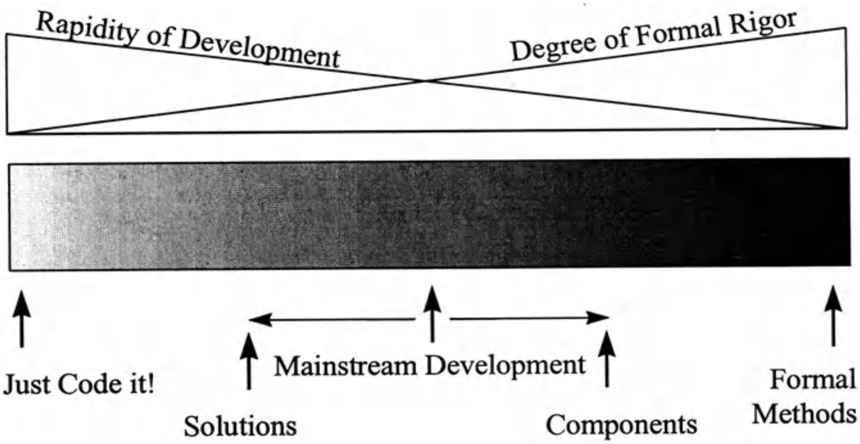


***Figure 1:** The Software Process Spectrum of Rigor*

2.2 Solutions and Components

Most mainstream development falls somewhere in the middle band of this spectrum. The means must justify the end. The bottom-line is to trade off risks and quality features in determining the rigor of process. Very roughly, to give the reader a broad idea, solutions can be distinguished from components as shown in table 1.

Solution	Component
Much of the functionality can be exposed through the user interface.	Often, the functionality cannot be exposed through the user interface.
Scope not a big issue.	Scope a potentially huge issue.
Business requirements are specific.	Business requirements are generic.
Low degree of potential commonality.	High degree of potential commonality.
There is a narrow target user community.	There is a wide target user community.
The application is not computationally complex.	The application may be computationally complex
There are few interfaces to other systems with little "knock on effect" requiring changes to other systems.	There are potentially many interfaces to other systems with "knock on effect" requiring changes to other systems.
Immediate business value.	Business value not always immediate.
Quality criteria not a major issue(with exception of usability).	Stringency of quality criteria (for example, performance and reliability)
Harvesting of re-use of legacy assets.	Sowing of re-use of legacy assets such as legacy systems, databases and software packages
Timeboxing is a viable strategy.	Timeboxing is not a viable strategy.

Table 1: *Solution Projects Versus Component Projects.*

2.3 The Delivery Process

Responsible RAD is essentially a delivery process that can take various forms, for example figure 1 illustrates a process which works well for object-oriented projects using established modelling techniques such as use cases [4]. Feasibility is essentially a scoping exercise which is expected to be completed once for a set of requirements. This may result in several "sub projects" for sets of use cases depending on the scale and complexity of the proposed solution. Analysis and prototyping are pursued iteratively to bring requirements into sharper focus. This results in identification of a number of use cases for which increments are to be designed in terms of user services. Each increment undergoes a design, acceptance and roll-out loop, with feedback iterating back to the planning of the next increment and to adjustment of analysis where necessary. Prototypes produced in analysis may be used to help drive the design, and may be evolved into the working increment. The delivery loop is repeated in iterative fashion so as to evolve the system in line with actual use.

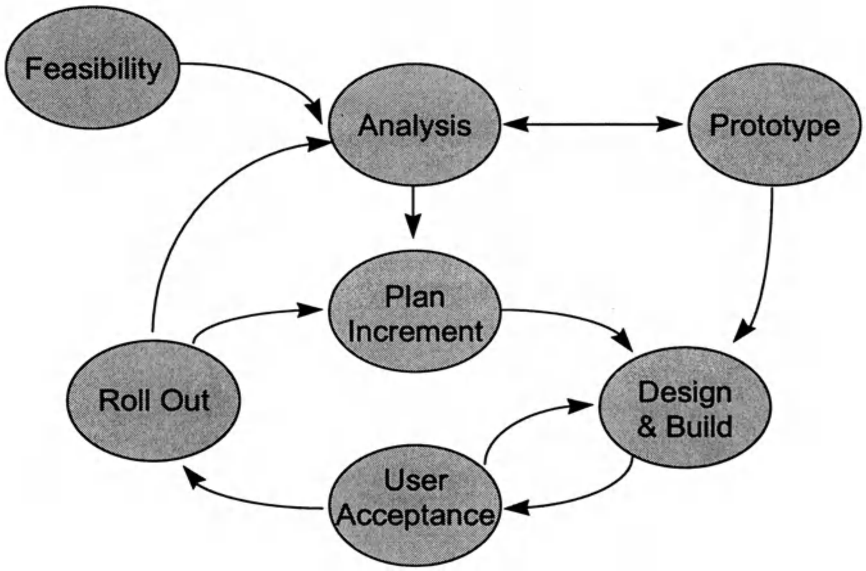


Figure 2: *The SELECT Perspective* Delivery Process

2.4 Component Challenges

Although it is recognised that costs will be higher and pay-back periods longer a component project should also be divisible into small increments that provide regular tangible results in response to business needs. It is the level of rigor and detail that is much greater with components, reflected in the techniques used, not the delivery mechanism. The iterative incremental approach also works best for component-based *delivery*. In fact most software delivery works naturally this way. A "big picture" provides a context. Chunks of the picture are then honed into sharper focus by working iteratively (a little analysis, a little design, a little implementation and round again). However, components include four further challenges:

- software re-use across different parts of the business
- potentially high risk factors
- demanding quality criteria
- the need to leverage legacy software.

These challenges call for more than a simple use case driven approach to modelling and a software process which provides control and co-ordination of projects.

2.5 The Waterfall Revisited

A common response to these challenges is another type of project that I see regularly in my work as a consultant. This type of project is often faced with integrating multiple legacy systems in with the new component technology. In contrast to the "no modelling" projects, these projects often get bogged down in the tar-pit of unnecessary modelling detail. It is argued that the only way to reach the standards of rigor demanded for reusable components is to model in detail producing separate logical and physical models across the enterprise.

It was no surprise to me to read the following survey result from Cutter Information: "Only 7% of IT organisations have completely abandoned the waterfall lifecycle approach to systems development, and 44% say they use it most or all of the time on their projects! By contrast only 29% say they use a RAD approach." [5] Too often elaborate theoretical models are produced which do not result in useful software applications. Traceability between separate logical and physical models becomes an administrative nightmare. Often, models are "thrown away" in the rush to meet deadlines in the face of management frustration at the modelling inertia associated with such approaches. This situation is exacerbated by the fact that most methods virtually ignore the need to leverage legacy systems and databases and package software. The major problem here is not only lack of the right architecture, it's often that there's no architecture at all.

3 The Role of Architecture

3.1 Service-Based Architecture

I use an architecture that helps to take a service-based view of software [6]. Services are accessed through a consistent interface that encapsulates the implementation. Each service has a published specification of interface and behavior. Components provide the interfaces. Services provide for a much more effective *molecular* level of reuse based upon software structures that transcend the individual *atomic* object. In contrast objects are usually too fine-grained to be useful by themselves, seldom standing alone in enterprise systems. In fact, providing its interfaces conform to standard, the component doesn't have to be written in objects at all.

3.2 Service Categories

A user service delivers business capability through the software interface to a business process. At a high level a user service is identified and developed using use case modeling. At a lower level a user service may be split using probes or operations into a set of smaller (more reusable) user services. A user service can call on business and data services via operation calls.

Business and data services are usually shared by different user services. A use case breaks down into a set of business service requests. Business services apply business rules in order to convert data information. Data services interact with physical data storage mechanisms and supply business services and user services with data they need in a technically neutral manner. Business and data services may be split using probes or operations into sets of smaller (more reusable) business and data services.

The service architecture works on the classic basis of gathering content (data), structuring and translating the content into form (information) and interpreting information (business capability) (see figure 3).

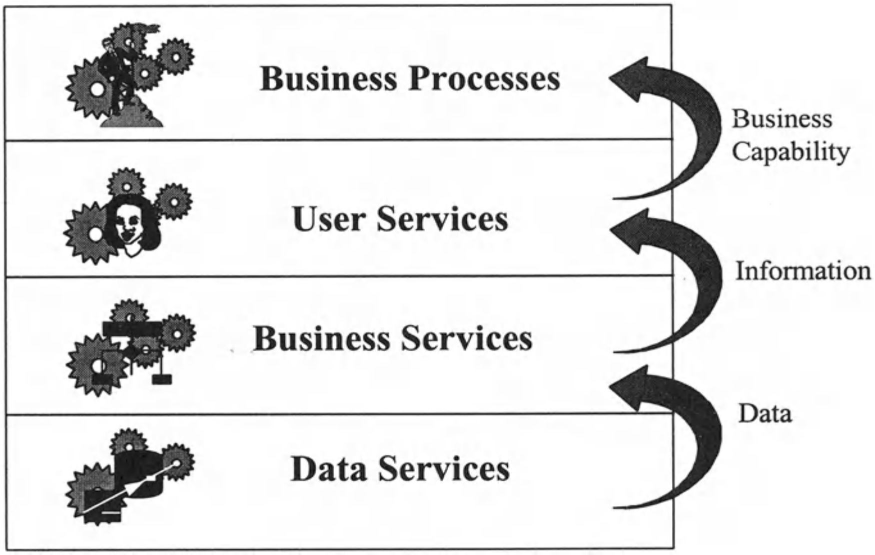


Figure 3: *From Data to Business Capability*

3.3 Design Philosophy

The service-based architecture provides a design philosophy for re-use of models at the high-end of the process, before any code is attempted, with correspondingly greater opportunities to re-use *business* logic and to increase software quality. Concepts from the standard Unified Modelling Language [7] are applied in a pragmatic fashion; [8] provides more detail.
This architecture also helps capitalise existing services into new offerings or products. Instead of starting each project every time with a "clean slate" sets of service features are examined to see which can be reused to solve a business problem. Software that is already proven to meet existing functional requirements, but often fails to meet non-functional requirements is re-engineered to meet those non-functional requirements using the new component technology.

For example, a recent project involved a dealing system in which various *user services* provided visible functionality in the form of sets of window-based GUI forms. User services included placing deals, authorising deals, setting up schemes, removing schemes and so on. A set of reusable *business services* supported the user services in their function. Business services provided by an existing COBOL system included checking ability to proceed with a deal and verifying security types. Other business services involved new functionality to provide sophisticated fund switching. These business services used a set of reusable *data services* to access the necessary fund data from a legacy database transparent to the user. A solution project targeted development of a new GUI system to provide the user services. Component projects focused on renovation of the COBOL security systems and legacy fund database plus a new set of fund switching business services in C++.

This way of looking at the process, illustrated in figure 4, also reflects the stabilisation of distributed object standards and the emergent component technologies which move us beyond the world of client-server into a world where the network is a living system which evolves in harmony with business needs. The Internet applies this concept globally: it is based on a world network that is more and more taking on the appearance of one gigantic computer. In combination these technologies open up the possibility of a *component marketplace* based on the consumer-supplier concept first defined by Brad Cox [9], who likened software development to the electronics industry where new solutions are assembled from integrated circuits.

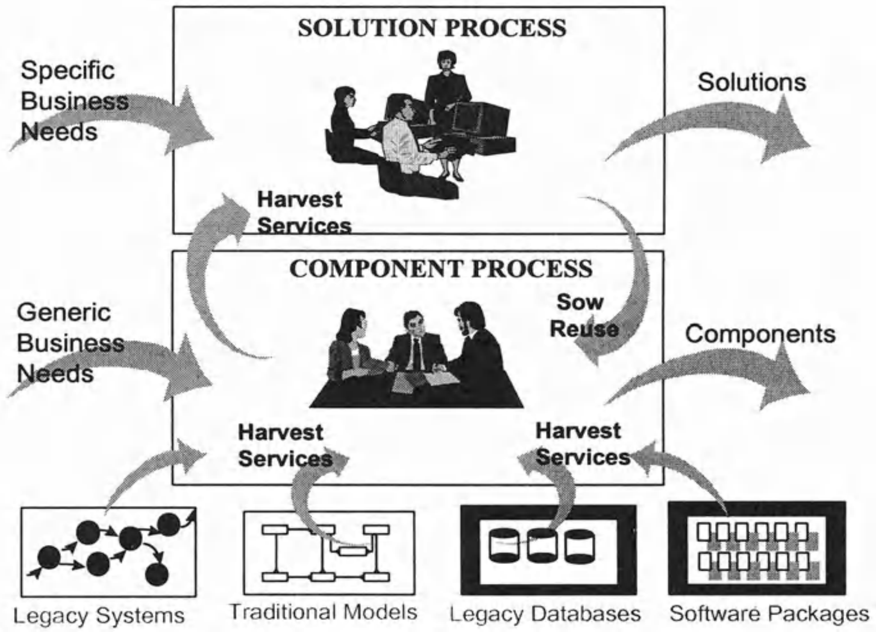


Figure 4: Solution and Component Processes

4 The Impact on Quality

4.1 Balancing Risk and Value.

Risks must be assessed against quality features. How exactly though do we "position" a process along the spectrum discussed earlier? The level of rigor appropriate for a particular development is dependent on perceived value of the software to be produced. For instance, if the main value driver is speed of delivery of a simple data capture or inquiry system with a short lifespan then a development process with little rigor may be appropriate. If, however, the software is for a safety critical flight management system then maximum reliability, accuracy and maintainability is required from the development process.

Positioning of a suitable process along the spectrum of rigor is of course much more complicated than described in the above simplistic scenario. For example, perceived value is a function of many different software and process quality features, which are traded off against acceptable risk. Software features include quality attributes such as performance, security and usability [10]. Process quality features include speed of delivery and effort involved [11]. Risks include user computer literacy and system volatility. A full discussion of trading off risks and quality features is outside our scope; the reader is referred to [12].

The salient point is that the appropriate degree of rigor in the development process should balance the risk accounted value drivers. For example, acceptable degree of rigor will be a function both of the speed with which early results are required and the level of reliability and maintainability required for those results. Often we find that processes are applied blindly without sufficient thought concerning perceived value. For example, much unreliable and unmaintainable software has resulted from first generation client/server developments working exclusively to speed of development and look and feel of user interface as their only criteria. Conversely other systems are over-engineered to the dictates of elaborate standards, resulting in systems which are never actually delivered because of very justified user frustration.

4.2 Component Team Roles

Team roles are an important part of the process. Solution team roles are already well-covered in the literature [2], so excluded here for reasons of scope. The component team roles include:

Re-use Manager: The re-use manager plans and controls the activities of the component team, makes policy decisions concerning re-use and has a watching brief for overseeing component projects.

Re-use Librarian: The re-use librarian has a combination of administration and technical skills. The role controls the generic models and repository and also publicises capabilities of the generic models and repository. It is responsible for checking in and checking out model items or components.

Re-use Assessor: The re-use assessor has good working knowledge of existing systems, packages, databases, generic models and available components. This knowledge is used to assess re-use opportunities (for example legacy systems and local models) and to evaluate re-use requirements. The role identifies areas for re-use improvement and pollinates re-use across solution projects.

Re-use Architect: The re-use architect has overall vision, carries out architectural scoping, identifies and acquires reusable components and carries out architectural modelling. The role promotes the value of reuse, acting as a "reuse promoter".

Component Developer: This role codes and tests components and calls for highly developed program design skills.

4.3 The Control Cycle

The major differentiator of the component process is that it includes a control cycle as illustrated in figure 5. Let's look at the stages of this control cycle, which are essentially proactive and ongoing:

Architectural Scoping: Sets of generic business requirements are analysed in terms of business services and/or data services and generic models. Non-functional requirements are analysed in terms of an overall quality plan. Proposed plans for services (emerging from the Plan Services stage) are gauged against the models, which are adjusted where necessary. This activity is typically performed by the re-use architect role.

Assessment: Assessment is a co-ordination activity which aims to pollinate re-use across solution projects. It involves identifying and weighing up requirements for services against the generic models. Potentially reusable services, may arise from analysing legacy assets or as a result of analysing feedback from either solution projects or roll out of components. Assessment also determines whether separate feasibility and analysis stages are required. This activity is typically performed by the re-use assessor role.

Plan Services: This includes identifying the type of project, and assessing risks against quality criteria. The plan should provide an overall indication of the order of delivery, the contents of each service and the estimated timescales and costs and a detailed plan for development of the next set of services. This activity is typically performed by the re-use manager role.

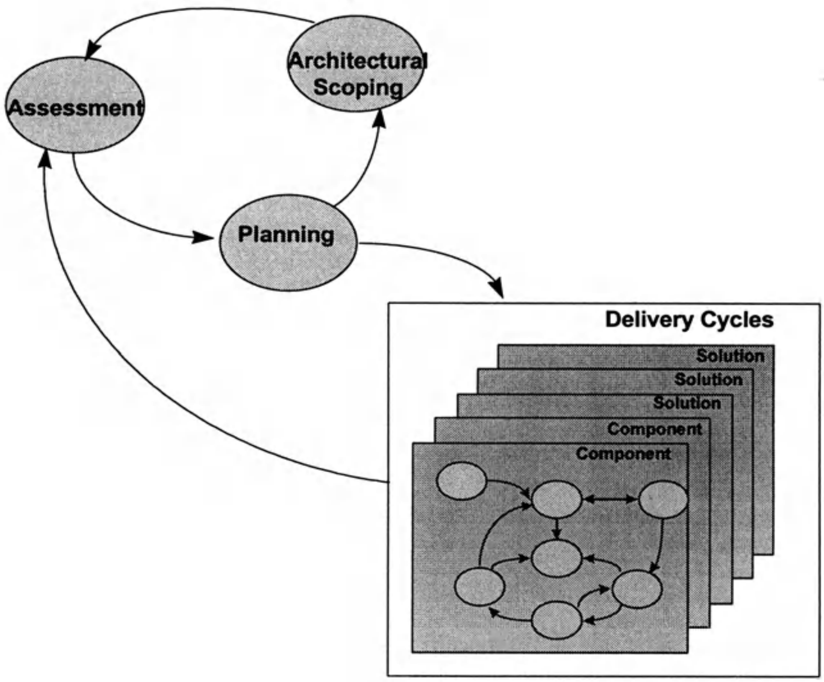


Figure 5: Control and Delivery Cycles

4.4 Component Management

The interplay between solution and component processes requires effective component management as illustrated in figure 6. Service-based component management tools provide the ability to browse, install and register the components in a repository. A good component management tool enables organisations to make their components available to a wide audience via the Internet. The tool provides facilities to make the use of the component marketplace easier, by allowing users to register interest in particular catalogs of components. The repository includes definitions of component interfaces and the services supplied through those interfaces. The repository should also enable identification of models that are associated with the components and that we compose, extend and adapt, through the modelling process. Also, in practice re-use is not a binary concept: a good component management tool should provide facilities for controlling and administering levels of re-use as described in [13]. This activity is typically performed by the re-use librarian role.

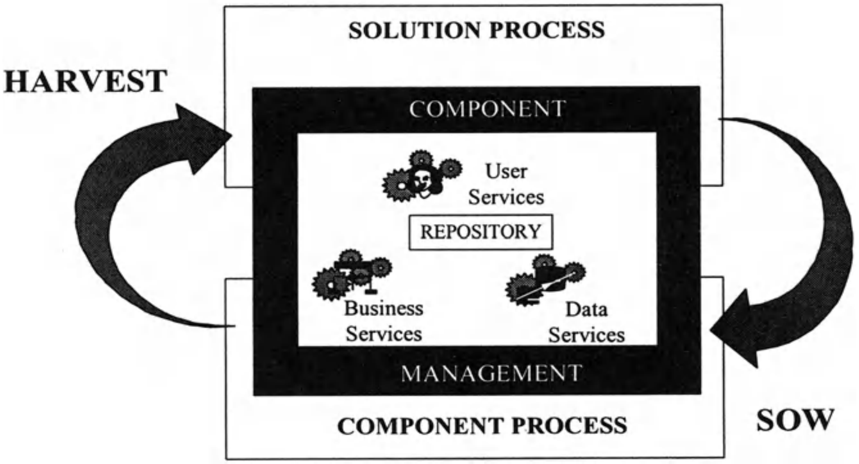


Figure 6: *A Service-Based Process*

5 An Example

The following simplified example gives a small taste of how modelling techniques, based on the Unified Modelling Language [7], are used in the control process.

5.1 Architectural Scoping

Techniques such as domain analysis are used to plan our software architecture in terms of *packages*. A package is "a general purpose mechanism for organising elements into groups" [7]. Such elements can range from model items (classes, use cases and so on) to legacy assets (for example, compiled code, transaction, or database libraries).

A package diagram is used below to scope architectural dependencies (shown as dashed arrows). Service categories are used to organise the packages (hence the term "service package"). Typically, service packages are allocated to different teams. Project management is facilitated by architecting service packages in early phases of development. This also has the advantage that incremental design can focus on specific implementation detail without being overloaded with wider architectural concerns. Services are released in incremental fashion through a set of evolving components.

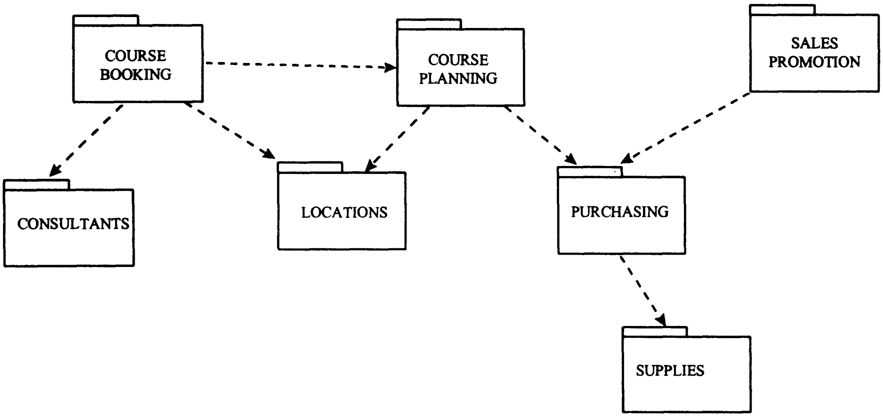


***Figure 7:** Example Package Diagram*

Class modelling is applied at enterprise level to understand the business domain. High level business classes are allocated to packages as illustrated in figure 8. The diagram is simply intended at this stage to provide a conceptual road-map.

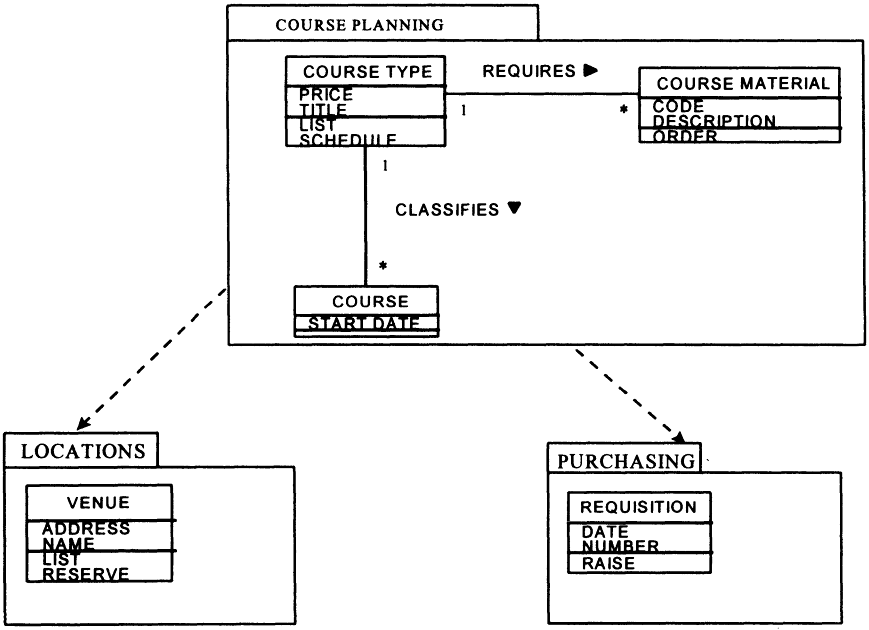
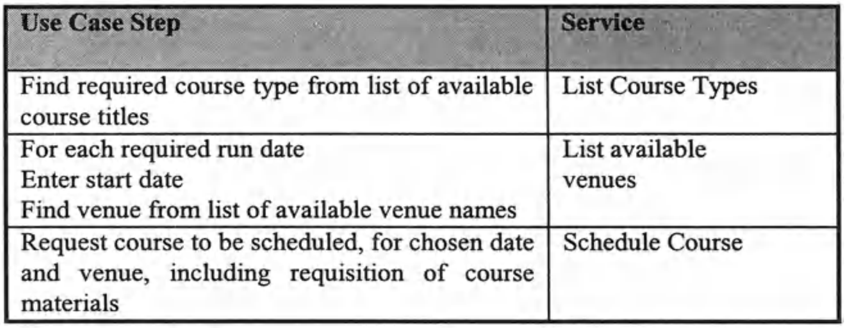


***Figure 8:** Example High Level Class Diagram with Packages*

5.2 Assessment

In assessment candidate legacy systems are evaluated for capability to provide services. In our example Locations and Purchasing are infrastructure processes. Locations business services are to be provided by a legacy system, Purchasing is to use an existing database to provide data services, Course Planning is a business process for which a new solution is to be developed to provide user services.

One of the use cases within Course Planning is Add Scheduled Course. A solution project analyses this use case in terms of service requests as shown in table 2. A component management tool is used to help search for the candidate services.



Use Case Step	Service
Find required course type from list of available course titles	List Course Types
For each required run date Enter start date Find venue from list of available venue names	List available venues
Request course to be scheduled, for chosen date and venue, including requisition of course materials	Schedule Course

Table 2: *Add Scheduled Course Use Case*

Assessment assists the solution team in exploring how objects identified in domain analysis might work together to provide services to support use cases. UML collaboration diagrams are very useful for this, as shown in figure 9 for the use case Add Scheduled Course. Note the UML path name notation to show parent packages. Collaboration diagrams are also useful in detailed analysis and design. Here their use is essentially exploratory. This technique is akin to CRC cards [14] and can be applied that way using a white board.

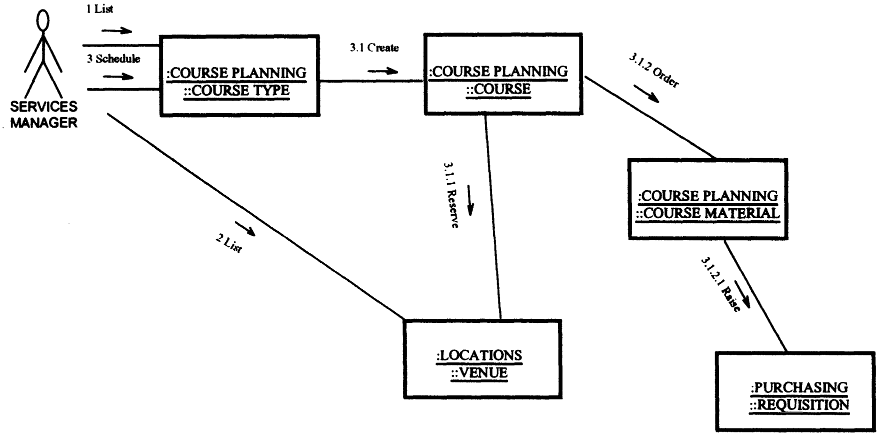

1 List
3 Schedule
SERVICES MANAGER
:COURSE PLANNING ::COURSE TYPE
3.1 Create
:COURSE PLANNING ::COURSE
3.1.2 Order
:COURSE PLANNING ::COURSE MATERIAL
3.1.1 Reserve
2 List
3.1.2.1 Raise
:LOCATIONS ::VENUE
:PURCHASING ::REQUISITION

Figure 9: Example Collaboration Diagram for Add Scheduled Course Use Case (Normal Scenario)

Obviously, I have only touched on many of the issues by way of illustration. As development unfolds so further UML diagrams are bought into play. Use cases are amongst the best documented of these techniques and are particularly useful in solution development. The layering features of sequence diagrams are less well generally understood but form an important vehicle for modelling the interplay between different service types. Full details of our approach are described in [8].

6 Conclusion

Much current literature covers only the application of RAD to delivery of solutions in isolation. Long term this is not conducive to producing good quality software, which is a key requirement of components. To address the potential conflict between RAD and quality I use a service-based architecture together with separate control and delivery processes in which acceptable risks are balanced against quality requirements. The value of modelling as a means of capturing functional requirements is well covered in the literature. Less well understood is the role of modelling at an architectural level. This helps expose quality issues early in the process and brings solutions and components together in integrated fashion.

References

1. Martin, J., *Rapid Application Development*, Macmillan, New York, 1991

2. DSDM Consortium, *DSDM Version 3,* Tesseract Publishing., 1997.

3. Stapleton, J., DSDM - The Method in Practice, Addison Wesley Longman, 1997

4. Jacobson, I., Christerson,M., Jonsson, P., Overgaard, G., *Object Oriented Software Engineering: A Use Case Driven Approach*, Addison-Wesley, 1992

5. Yourdon, E.N, *The Yourdon Report*, May, 1997

6. Mowbray, T.J., and Malveau, R.C., CORBA Design Patterns, Wiley, 1997

7. OMG, *Unified Modelling Language Version 1.0,* OMG, Framlington, Mass.,1997

8. Allen,P. and Frost,S., Component-Based Development for Enterprise Systems, Cambridge University Press- SIGS, 1998

9. Cox, B., *Object-Oriented Programming: An Evolutionary Approach*, Addison Wesley, 1986

10. Gilb,T., *Principles of Software Engineering Management*, Addison-Wesley, 1988

11. Fenton, N.E., *Software Metrics*, Chapman and Hall, 1991

12. Folkes,S., and Stubenvoll,S., *Accelerated Systems Development*, Prentice Hall, 1992

13. Allen,P. and Frost,S., Component Manager, *Select Software Tools White Paper*, 1996

14. Wilkinson, N.M, *Using CRC Cards: An Informal Approach to Object Oriented Development*, SIGS Books, 1995

Transient Quality performance within the 'SPAN' of new Software Development Process and Technology establishment

Nikos Lykouropoulos, Ph.D., Andreas Maniatis, M.Sc
I.M.F. Ltd., Athens, Greece

1 Introduction

As the necessity for formal process and quality assessment standards is becoming more and more accepted, the market position of a software producing organisation tends to be determined by its quality performance. Driven by this impetus, considerable effort has been assigned by a major percentage of the software producing organisations within the framework of process improvement initiatives.

The establishment of new process elements resulted into important organisational and labour changes. The introduction of these new process elements may last from a few months to several years, depending on the current status and the objectives of the organisations. During this period, the performance of the organisations is affected both in efficiency and in quality. Although the economical parameters of this disturbance can only be evaluated within the context of each specific organisation, they are always interpreted into an increase of cost.

The size of the necessary investment for process improvement can be approximated referring to published case studies and available research results from the government and the private sector. However, the risk, the performance degradation, and the extra cost due to the adjustment period cannot be safely estimated. The reasons for that are lack of appropriate data and significant variance in the evolution of the phenomenon between apparently similar cases.

2 Process improvement within SPAN

During the last twelve months, IMF Ltd. has put on a great effort in order to improve its process, methodology and technology. After an internal assessment (an informal one) the company has been placed at the Initial Level of CMM. For this reason it was decided to initiate a Process Improvement Effort along with the simultaneous migration to new technologies and methodologies. This effort is funded by an ESPRIT-ESSI PIE (SPAN-23896).

The company developed - and has already assimilated - a new iterative life cycle model. This model is based on Boehm's Spiral Model [1] and embeds multiple, priority-driven, iteration levels. The main concept is the dynamic definition of incremental deliveries based on the evaluation of quality parameters. From the methodologies-technologies point of view, the notions of Use Cases, Object Oriented Analysis and Design (OOAD), and DCOM-CORBA component

C. Hawkins et al. (eds.), *Software Quality Management VI*

technologies have been introduced. This new framework is being evaluated through its application on two case studies comprising of normal company business.

3 An Extended Quality Model

Quality estimation is based on quality models that decompose quality perception into quality factors (which - in the general case - are external attributes of the product). Nearly all available quality models deal only with the end product of software development; that is the delivered system. However, software production is also a service and a complete quality model must embody the service aspects as well. Total cost and time of delivery are the most obvious parameters of customer's concern. The scheduled time for first operation is very critical to many applications. Apart from the development cost, delays in delivery are often evaluated by the customer as cost increments as well.

IMF has developed a quality model that corresponds to the Service aspects of its software development process. The overall quality model is based on McCall's software quality model [2]. The primarily identified service-related quality factors are *On Time*, *In Budget*, *Observability*, *Controllability* and *Credit*.

4 Results and Lesson Learned

The process improvement experiment is divided into two phases. The first phase is dedicated to the preparation and infrastructure establishment, as well as to the training of the employees into the new process and OOAD principles. The second phase is dedicated to the development of two products using the new process. The first project is a 'system service' exploiting distributed components and the other is a typical 'data intensive' application developed using RAD technology. Metrics defined into the new process are also applied on a third IMF's typical application, developed using the previous company process.

The results can be reflected on the following conclusions: 1) The use of small iterations (around to ten development days) result into less acceptance test defects - an improvement of around 50%. 2) With the use of the new process there are more, but with better overall effect, schedule revisions. 3) The use of Object Orientation can degrade estimation accuracy in non compatible software (e.g. system software) 4) There is a decline in company's productivity performance during the period of extensive training and the introduction of the new process and technology. If the introduction of new technology is - for some reason - urgent, then better performance can be achieved by postponing the process improvement till the completed and thorough assimilation of the new technology.

5 References

1. Boehm, B.W., A Spiral Model for Software Development and Enhancement, IEEE Computer, May 1989, pp. 61-72
2. McCall, J.A., Richards, P.K., and Walters, G.F., Factors in Software Quality, RADC TR-77-369, 1977. Vols I, II, III, US Rome Air Development Center Reports.

Section 3

Object Oriented Issues

Providing Support for the Integration of Structured Methods and Quality Management Systems

J Barrie Thompson, Helen M Edwards and Colin J Hardy
Software Engineering Group,
School of Computing and Information Systems, University of Sunderland, UK

Abstract

An appraisal of the state of the software industry with regard to methods and quality is presented which highlights the fact that methods are commonly modified in an attempt to fit specific situations. The possible dangers in approaching this in an informal manner are detailed and areas of concern which the industry needs to address are proposed. The actions that should be taken to ensure that quality is achieved are detailed and a method, MEWSIC (Method Engineering With Stakeholder Input and Collaboration), is outlined whose aim is to formalise the development of situational methods so that links to quality assurance processes can be retained. A high level overview of MEWSIC is given and then detail of the two stages in the method is given, bringing out the collaborative nature of the approach. Finally an appraisal of the MEWSIC method and its sphere of application is presented.

1. Introduction

The software industry, when considering quality, has tended to concentrate on two aspects: the product itself (the software) and the processes involved in producing that product. Assessing quality is fraught with difficulties since not only is the final product - the executing code- intangible, but as many authors have pointed out (e.g. [1]) quality itself is hard to define and almost impossible to measure directly. Thus the industry has tended to concentrate on the production process and use the argument that:

C. Hawkins et al. (eds.), *Software Quality Management VI*

"a quality process should lead to the production of a quality product" [2].

If this argument is accepted, the importance of a quality management system (QMS) for the production of software and the accreditation of the QMS with regard to a quality management standard becomes very clear. It is also obvious that any QMS for software production must depend on well defined production processes. These will be best provided by well defined and rigorous approach(es) such as structured method(s). The above argument can be rewritten as:

"We can have confidence in the software (the product) if we employ a well defined structured method whose use is quality assured within an accredited quality management system." [2]

It is thus very clear that the ways in which methods are actually used in practice can greatly affect the quality of the software.

In section two of this paper we present an appraisal of the software industry with regard to methods and quality which highlights the fact that methods are commonly modified in an attempt to fit specific situations. The possible dangers in approaching this in an informal manner are detailed and areas of concern which the industry needs to address are also proposed. In section three the actions that should be taken to ensure that quality is achieved are detailed and a method, MEWSIC (Method Engineering With Stakeholder Input and Collaboration), is outlined whose aim is to formalise the development of situational methods so that links to quality assurance processes can be retained. A high level overview of MEWSIC is given in section four and then details of the two stages in the method are given in sections five and six. These bring out the collaborative nature of the approach. Finally, in section seven, an appraisal of the MEWSIC method and its sphere of application is presented.

2 The State of the Software Industry

The public perception of IT as frequently promoted in advertisements on television and in the press is that it is an industry of excellence where all is possible now, or in just a few years time. In fact, the real state of the UK industry is very different. The actual performance of IT was been summarised at the start of 1996 as [3]:

> "80 - 90% of IT investments do not meet their performance objectives; the reasons for this are rarely purely technical in origin.
>
> Around 80% of new systems are delivered late and over budget.

> Most organisations are not good at evaluating the performance and impact of their investments in IT"

These statements represented a quantitative summary of the views of 45 major researchers and consultants in the UK drawing on results from around 14,000 organisations. A slightly brighter picture of an industry which continues to improve, if somewhat slowly, can be obtained from the results of a number of surveys undertaken by researchers within the Software Engineering Group at the University of Sunderland which were reported at SQM 96 [2]. What these surveys show is that (over the last ten years) there has been a continuing increase in the use of methods and development tools and that software quality is on the agenda within many companies. Nevertheless, this survey and related work has also shown that often where organisations have invested in methods these have proved less than ideal [,4,5], these results are in agreement with those of others, for instance [6,7]. However, in many cases organisations have built up development infrastructures around these methods and have invested significant resources in trying to make them work.

One trend that has become clear is the use of what can be classed as "in-house" methods rather than rigorous adherence to named methods, such as SSADM, [8]. Coupled with this is a desire by organisations to customise their methods so that they will match particular situations. However, once developers decide that complete adherence to a specific method is not necessary then the advantages that practitioners and project managers see in methods are in danger of being eroded. For instance, it is common for organisations with formal quality assurance procedures and certification to have the use of a particular method written into its QA system. Once the link with the "approved" method is broken then the claim that work is carried out to approved standards is more difficult to sustain.

There are, therefore, a number of questions which the industry must address if it wishes to employ situational methods but retain the two arguments relating to quality, highlighted in section 1. These questions, as we highlighted in 1996 [2], are:

- How well specified are current methods as applied in practice?
- How appropriate are their frameworks for supporting a QMS?
- Where organisations adapt methods for particular projects, is this process engineered?

How is the engineering of a method formally incorporated into a QMS system?

3. A Solution

A solution to the questions proposed above is that each individual organisation needs to have a formal mechanism for method adaptation which can be rigorously examined within their QMS. In the academic environment method adaptation is normally classified as an area of method engineering (see for example [9,10,11]). This area of software engineering typically concentrates on the technical issues involved rather than on the practitioner aspects such as management and quality. To make the research developments in the field of method engineering accessible to practitioners in the field we believe there is a need to provide a methodological approach that can be explicitly followed and can itself be quality assured. This process can then be used with confidence to develop situationally specific methods. To ensure that the method developed is appropriate we also believe that such a process must identify the stakeholders in the project whose concerns will impact upon the type of method required for development.

The method which we are proposing, MEWSIC (Method Engineering With Stakeholder Input and Collaboration) encapsulates the two key features that need to be considered by anyone concerned with formalising the development of situational methods: these are the stakeholder input and the method engineering process. Situational methods satisfy a real world desire to move from "cookbook" approaches to appropriate methods. However, there is often a link in organisations between their prescribed development methods and their quality assurance processes and it is important that this link is not lost or weakened. Uncontrolled method tailoring runs the risk of allowing this to happen, but a formalised approach to the development of situational methods allows this link to quality processes to be retained.

The second key feature to be addressed in answering the need for situational methods is in adequately accounting for the number of stakeholders who have a legitimate interest in the success of the project. Since these stakeholders have different interests, concerns and skills it is necessary to distinguish between those who provide input that informs the method engineering process and those who carry out this process.

4. Overview of the MEWSIC Process

The purpose of MEWSIC is to provide software developers with a practical approach which will ensure that decisions taken in the method engineering process are recognised, justified and are capable of being audited. The approach has two main stages:

1. Derive Situationally Relevant Factors and
2. Define the Method for Use.

Within these stages activities and deliverables are specified that provide a mechanism by which the required development method can be defined and justified. This approach is adopted to facilitate quality assurance, since the formalised manner which is adopted is amenable to auditing and objective analysis. Such an approach also enhances the ability of an organisation to replicate activities, be aware that a specific strategy has been developed and identify those decisions that have led to high or poor quality results. In both stages of MEWSIC collaborative work is undertaken: in the first stage the collaboration is with all stakeholders who are affected by, or who affect, the planned systems development project. In the second stage the collaboration is at a different level since it is concerned with the engineering of the required method. Therefore it relies on the technical skills and expertise of those who are involved in the development and planning of software projects (typically, project managers, team leaders and software engineers).

The first stage is iterative in nature and uses the techniques and concepts associated with soft systems approaches and participative work groups (as used for instance by Checkland [12] and Mumford [13]). This first stage involves work by two teams of people: a technical team whom we term "Method Design Team (MDT)" and a larger group representing all stakeholders the "Collaborative Stakeholder Group (CSG)". The identification of stakeholders can assist in the derivation of areas of concern, or conversely acknowledgement of areas of concern can lead to the detection of previously omitted stakeholders. The output from this stage is an definition of stakeholders for the specific project (and their relative importance) and a mapping of the prioritised concerns to these stakeholders. This first stage is discussed in more detail in section 5.

Once this primary stage has been completed the second stage can begin. It is in this stage that the systems development method is engineered by the MDT and is based on the concerns that have been identified by the stakeholders. For many organisations this stage will begin with an examination of their existing methods to determine if any of these is suitable and whether changes need to be made. However, this stage is flexible enough to allow the method design team to construct a method from individual components - if they have the competencies so to do. (It is worth noting that this is an example of an "area of concern" which should have been addressed in the first stage of the MEWSIC process). Again, during this second (method engineering) stage the decisions which are made and the reasoning supporting them would be explicitly recorded. This second stage is discussed in more detail in section 6.

5. MEWSIC Stage 1: Derive Situationally Relevant Factors

This stage has five main tasks within it:
1-1. Identify Initial Stakeholders.
1-2. Identify Initial Scope of the Project
1-3. Identify the Areas of Concern and Additional Stakeholders
1-4. Prioritise Areas of Concern and Stakeholder Importance
1-5. Map the Areas of Concerns to Stakeholders
These are shown diagrammatically in Figure 1.

In Step 1-1 "Identify Initial Stakeholders" the Method Design Team (MDT) use the project initiation document (or project scoping document) and any existing template of stakeholder definitions to define those stakeholders (not individuals) who are expected to be affected by or affect the project: and thus the influence the design approach used. Typical stakeholders are, for instance: the project sponsor, the project manager, the team leader for the development team, the end user, the operations manager. Once these stakeholders have been identified a representative is nominated to contribute to the "Collaborative Stakeholder Group" (CSG), who are involved in steps 1-3 to 1-5. This group includes the method design team as they are also stakeholders in the process. Step 1-2 is carried out in parallel with Step 1-1, in this step the document that initiates a project (such as a project initiation document or project scoping report) is examined to identify the general feature that are of relevance to the project and its development approach. Such features would include, for instance, project size, complexity and importance to the business.

Once these initial two steps have been carried out the remainder of the stage is undertaken by those who make up the Collaborative Stakeholder Group. Although the activities are broken into steps and the basic sequential flow is shown in Figure 1 it would be expected that these would be carried out iteratively. In Step 1-3 the areas of concern within the project are identified using approaches such as participative work groups to identify the areas of concern that exist for each set of stakeholders and any additional stakeholders that are revealed as a result of this process. The outcome of the step is a list of areas of concern for specific stakeholders: these areas of concern are analysed to ensure that equivalent concerns from different groups are matched up. In Step 1-4 the CSG then move on to prioritise the areas of concern, order the stakeholders in terms of importance for the project and produce a mapping of areas of concern to stakeholders whose topology reveals the range of areas and level of concern. It is this mapping that is used as input into the second stage of the method.

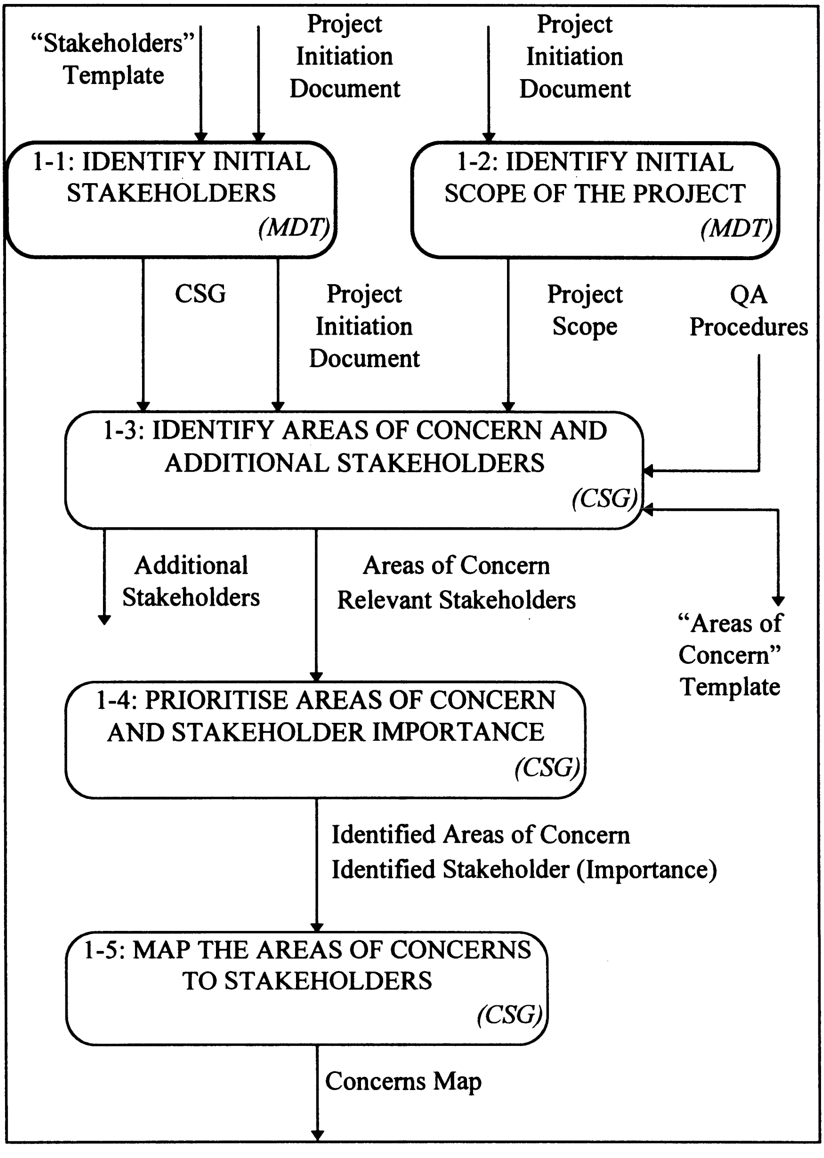

"Stakeholders" Template
Project Initiation Document
Project Initiation Document
1-1: IDENTIFY INITIAL STAKEHOLDERS (MDT)
1-2: IDENTIFY INITIAL SCOPE OF THE PROJECT (MDT)
CSG
Project Initiation Document
Project Scope
QA Procedures
1-3: IDENTIFY AREAS OF CONCERN AND ADDITIONAL STAKEHOLDERS (CSG)
Additional Stakeholders
Areas of Concern Relevant Stakeholders
"Areas of Concern" Template
1-4: PRIORITISE AREAS OF CONCERN AND STAKEHOLDER IMPORTANCE (CSG)
Identified Areas of Concern Identified Stakeholder (Importance)
1-5: MAP THE AREAS OF CONCERNS TO STAKEHOLDERS (CSG)
Concerns Map

Figure 1: The Steps of Stage 1: Derive Situationally Relevant Factors

6. MEWSIC Stage 2: Define the Method

In this stage the participants involved are limited to the method design team (MDT). This team is typically composed of:

- the project manager: this stakeholder has responsibility for the budget, the allocation of resources, ensuring conformance to QA standards, and the final deliverable (the live system).
- the project team leader: this stakeholder has responsibility for the specific project, the management of the development process, the organisation of the team, the operational issues affecting the team, and conformance to QA standards.
- the software engineer: this stakeholder has responsibility for the development of elements of the specific project, possesses expertise in certain development approaches, is responsible for achieving his/her set task deliverables.

The stage consists of two steps:
2-1. Design the situational method
2-2. Validate and Verify the Situational Method.

The main activity is carried out within the Step 2-1. The method design team have as inputs to this step:

- The prioritised areas of concern and list of stakeholders for this project
- The "Concerns" map for this project
- The definitions of the methods, techniques and development tools available to the development team.
- Historical "Concerns" map for previous projects and the associated situational method.

With these inputs the team can begin the method engineering process. This process is particularly dependent upon the relationships between these stakeholders and the method aspects. The method engineering activity can be complex and is based upon the approach discussed in [11]. More details, including examples can be found in [14].

The second step in the stage is needed to ensure that the proposed situational method actually matches the "concerns" map from Stage 1. If this is acceptable then the process is complete and the system development activity can begin in earnest. If, however, there is a mismatch then the MDT needs to provide an analysis of the mismatch and identify to which part of the MEWSIC process they need to return.

7. Appraisal

The main features of MEWSIC are that it formalises the development of situational methods and explicitly focuses on the method engineering process and stakeholder input and collaboration. Since these stakeholders have different interests, concerns and skills MEWSIC distinguishes between those who provide input that informs the method engineering process and those who carry out this process. A primary feature for those concerned with QA is that the formalised approach ensures that the link between development methods and quality assurance processes is retained. However, it also offers the following additional benefits.

7.1 An Explicit Audit Trail

The use of a formalised method with specified tasks and identified deliverables ensures that an audit trail exists. This trail reveals: the decisions that have been taken (with the supporting justification), the concerns that have been articulated and defended. It thus reveals how the method used for a project was derived and judged to fit the required system and its development. The verification and validation step in the method provides confidence in the fit of the method to the project and its areas of concern. Therefore, this mechanism also increases the likelihood that such decisions could be justified and repeated for another similar exercise.

7.2 Knowledge Sharing

In these days of impermanence and change in working patterns, knowledge sharing and technologically assisted methods of collaborative working are essential for the corporate being, if not the individual. However, there are advantages for the individual also in that novices can be trained more effectively and that knowledge can be transferred even among experienced staff (since it is rare for one person to hold all the skills required within an organisation's software department). The articulation and justification of decisions that has to be made during the MEWSIC process enables software development teams and their stakeholders to share knowledge. In the technical area this is particularly important as it facilitates the learning and acquisition of skills from experienced personnel (such as project team leaders) to more junior systems developers. This addresses the problem that has been reported to us, in our empirical data, by software developers whose ability to customise methods is gained through personal experience. The MEWSIC approach of explicitly acknowledging the issues that are used to design a method helps to unearth the tacit knowledge that is often held by experienced software engineers.

7.3 Ownership

The two stage approach of MEWSIC encourages all stakeholders to "buy into" the system design process and feel ownership for the project. Therefore, the progress and success of the project is a common concern. This reflects the advantages of any participative approach, many of which are used in systems development, but it broadens the participation and ownership from a focus on the product to include the development of that product.

The future work on the MEWSIC method will concentrate on the further definition of the detailed tasks within the method and the testing of the method at case study sites. Some work has already been undertaken to develop the repository that is needed to support the method and this will continue, particularly focusing on the information retrieval and presentation issues.

Finally, for those organisations concerned with progress along the CMM levels, the MEWSIC approach offers one tool to help in continually improving their development processes by providing synergy between an individual development and the method supporting it.

References

[1] Kitchenham, B. Towards a Constructive Quality Model (COQUAMO) Pt. I and II, IEE Software Engineering Journal, 1987, Vol 2. No 4.

[2] Thompson, J.B, Hardy,C.J and Edwards,H.M. Structured Methods: Helping Software Quality or Raising Problems. International Conference on Software Quality Management, SQM96.Cambridge .April.1996.pp449-459.

[3] Clegg,C et al. The Performance of Information Technology and the Role of Human and Organisational Factors. Report to the Economic and Social Research Council, Swindon, UK., January 1996.

[4] Hardy, C.J., Thompson, J.B., Edwards, H.M. The Use, Limitations and Customisation of Structured Systems Development Methods in the United Kingdom, Information and Software Technology, Vol 37, No. 9, p.467, 1995.

[5] Hardy, Colin J., Thompson, Barrie J., Edwards, Helen M. Problems associated with the customisation of structured methods in the UK, BCS Conference on Information Systems Methodologies, Wrexham. Also presented at The Nineteenth Annual International Computer Software and Application Conference, COMPSAC '95, Dallas, 1995

[6] Dekleva, S.M. The Influence of the Information Systems Development Approach on Maintenance,MIS Quarterly, September, pp355-372, 1992.

[7] Palvia, P. Nosek, J.T. A Field Examination of System Life Cycle Techniques and Methodologies, Information and Management, 25, 73-84, 1993.

[8] Fitzgerald, B. The Use of Systems Development Methodologies in Practice: A Field Study. Information Systems Journal, July 1997, Vol 7 No 3, pp201-212.

[9] Harmsen, F, Brinkkemper, S, Oei, H. "Situational Method Engineering for Information System Projects", In: Olle, T.W., A.A. Verijn-Stuart (Eds.) Proceedings of the IFIP WG8.1 Working Conference CRIS'94 Maastricht, 1994, pp. 169-194

[10] Punter, T and Lemmen, K. "The MEMA-model: towards a new approach for Method". Engineering, Information and Software Technology, 38, 295-305, 1996.

[11] Hardy, C.J, Edwards, H.M and Thompson, J.B. "The Unification of Method Engineering Approaches", 4th Conference, BCS Information Systems Methodologies Group, pp. 439-450, 1996.

[12] Checkland, P., Scholes, J. Soft Systems Methodology in action, Wiley, 1990.

[13] Mumford, E. Effective Requirements Analysis and Systems Design: The ETHICS Method, MacMillan, Basingstoke. 1995

[14] Edwards, H.M, Thompson, J.B and Hardy, C.J. Developing Situationally Specific Methods through Stakeholder Collaboration. Submitted to IEEE compsac98.Vienna, August1998.

"Can You Have It All?": Managing The Time And Budget Against Quality Issue in A Dynamic Business Object Architecture Development

KSY HUNG[1], AJH SIMONS[2] and A ROSE[3]
[1,2] Department of Computer Science, The University of Sheffield
Regent Court, 211 Portobello Street, Sheffield, S1 4DP, UK
e-mail: [1] k.hung@dcs.shef.ac.uk ; [2] a.simons@dcs.shef.ac.uk
[3] CAD Consultants Ltd.
797 London Road, Thornton Heath, Surrey, CR7 6XA, UK
e-mail: [3] tony.rose@btrinc.com

ABSTRACT

It has been widely observed in the information technology (IT) communities that IT developers are coming increasingly under more pressure than ever in juggling between software quality and timely delivery in a tight budget. Developers are torn between the dilemma of either delivering quality software at a price of longer development time and higher cost or delivering software in a timely fashion neglecting quality. Many attempts have been made to tackle the challenge of: "Can-We-Have-It-All?". This paper recommends an approach to manage time and budget against quality and aims at achieving this tripartite objective. The paper covers the development of a Dynamic Business Object Architecture (DBOA) and its implementation through an insurance project case study. The structure and approach of the DBOA are explained through the development process and the case study is presented to demonstrate the initial result of this approach. Some insights resulting from applying the above techniques are also discussed.

1. Introduction

In the competitive business environment, software technology now plays a crucial role in providing increased lead time (shorter time to market). Electronic Information Systems (EIS) have become an integral part in most organisations [18]. It is a common goal amongst business organisations to improve the quality of, and gain added value from, their information processing application for a minimum investment in time and cost. In respect to the software quality issue, Object-Orientation (OO) is currently regarded as a better alternative for developing quality software than conventional structured methods [5]. OO promises, with its focus on encapsulated components, better maintainability, extensibility, scalability, portability and reusability. Furthermore, Business object technology has been developed to capture and define a model of the user's business and its information processing requirements [9,12]. A Business object is a coarse-grained object abstraction that encapsulates a typical, generic business task, adapted for a particular

C. Hawkins et al. (eds.), *Software Quality Management VI*

business domain [6,1]. Business objects are incorporated in a Business Object Architecture (BOA) [2], which is a re-configurable framework for handling the communications amongst business objects within a business domain. However, those techniques only swing the pendulum towards improving software quality regardless of the cost and time.

In regards to time and budget, Rapid Application Development (RAD) [11] is increasingly being adopted as the best way to deliver systems quickly and at low cost in many situations. The downside of RAD is that the pendulum swings towards shorter development time and reduced costs, at the expense of quality. As a result, software may be bug-ridden, or poorly structured which makes it difficult to maintain. The Dynamic Systems Development Method (DSDM) [3,15] has been adopted to impose a software life cycle on RAD and improve project management. However, this does not fully address the core issue of software quality, which is more affected by the development techniques used to capture and model business information.

This paper presents an approach that combines BOA and DSDM to provide a "Dynamic Business Object Architecture (DBOA)" [7] to develop quality software applications within practical time scales and for minimum costs. Projects using BOA techniques are implemented within the DSDM life-cycle environment. A credit insurance project case study was carried out using the DBOA approach to look at areas involving time and budget against quality. The structure of the rest of the paper is organised as follows. Section 2 outlines how Business Objects provide a mechanism to assist IT developers to understand the business domain better. Section 3 describes the adoption of a Business Object Architecture (BOA) to manage complexity encountered when reusing business-related software components. Section 4 describes the development of a Business Object Repository / Reuse Library to enhance the reusability of Business Objects. Section 5 addresses the necessity to involve business end-users to ensure the effectiveness and usefulness of the BOA to the business. Section 6 presents a DBOA framework applying the BOA coupled with DSDM life-cycle implemented through an insurance project case study. Section 7 evaluates the outcome from the case study. Section 8 justifies the balance between time and budget against quality by using a 'S.M.A.R.T.' evaluation criteria framework to evaluate the DBOA in terms of its 'Scalability', 'Measurability', 'Achitevability', 'Reusability' and 'Time-manageability'. And finally, section 9 concludes this paper and outlines further research work.

2. Business Objects And Business Object Architecture

2.1 What is a Business Object ?

Object Management Group (OMG)'s definition of Business Objects as: "*... a representation of a thing active in the business domain, including at least its business name and definition, attributes, behaviours, relationships, rules, policies, and constraints. A business object may represent, for example, a person, place, event, business process, or concept. Typical examples of business objects are: employee, product, invoice and payment ...*" [16]. Business objects can be viewed as Modelling Objects, used in the design process and as objects in the information system as illustrated in Figure 1. IT developers extract that 'modelling object' from the business and transform it into software components structured in Object-Oriented (OO) style, so that the modelled information can directly reflect the shape of the business model.

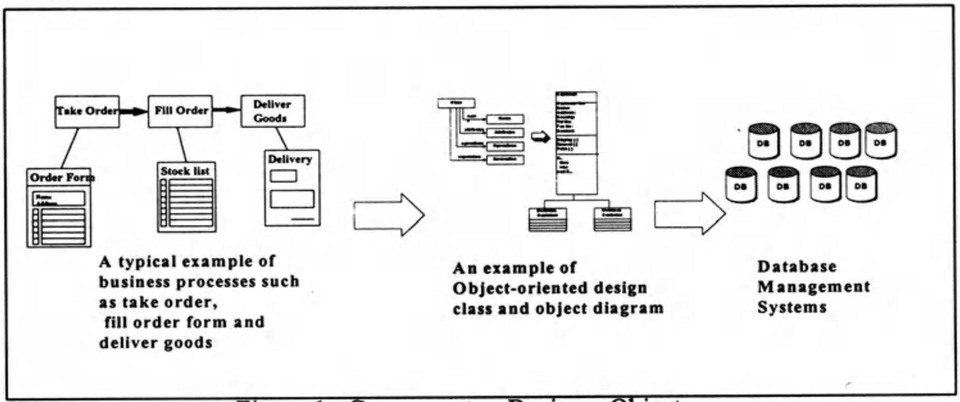

Take Order
Fill Order
Deliver Goods
Order Form
Stock list
Delivery
A typical example of business processes such as take order, fill order form and deliver goods
An example of Object-oriented design class and object diagram
Database Management Systems
DB

Figure 1 : Components a Business Object

2.2 Development of a Business Object

Standards are currently being developed by the OMG Business Object Domain Task Force through a request for proposals (RFP) from the software developers and academic researchers. The RFP is still under review pending for agreed common standard. At the moment there is no standard approach or method for creating business objects. To meet this need, we have combined Jacobson's Use Case Engineering (UCE) [9,10] and Ramackers' Domain Business Object modelling approach [13], as shown in Figure 2.

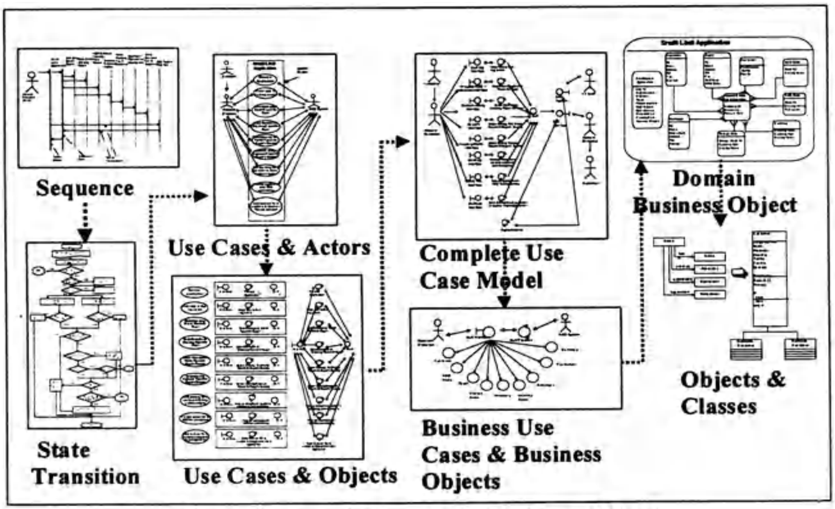


Figure 2 : Development of a Business Object

We use the sequence diagram, which gives an overview of the business processes; and the state transition diagram to describe the business processes step-by-step. The use cases and actors bring out the names of the tasks and how they are performed. These use case and object diagrams convert data into 'entity objects', processes into 'control objects' and actors into 'interface objects', these three different kinds of objects playing different roles. Since UCE does not provide any further techniques after the business use cases have been converted to business objects, we have adopted Ramackers' Domain Business Object to visualise all the components and relationships within a business object. Then we move on to object and class diagrams for implementation.

3. Business Object Architecture

3.1 Problems of Business Objects

We discovered the need for a different approach to modelling business objects after experience with using the UCE approach alone. We found that Jacobson's method did not produce business objects that were flexible enough for reuse in different contexts: the use-case driven approach tended to fix the interfaces of business objects; and the objects tended to be too coarse-grained, encapsulating business data and business processes in ways that were hard to break apart. In reality, businesses always face a situation in which different business operations might share common processes, tasks and data. If we develop coarse-grained business objects one after the other and put them together within a business domain, we will end up with object redundancy / object overlapping. For example, when we want to perform some business transactions such as invoicing and insurance claims, we need to involve customers. Does it mean that we need to include the customer object in both the invoicing business object and insurance claims business object? If so, we are overlapping the customer object. If not, how do we share the same business processes and objects? Moreover, we also need workflow direction to describe the sequencing of different business transactions, and the way in which these sequences may be broken apart, adapted and reused.

3.2 What is an Architecture?

The problems of Business Objects have prompted the adoption of a business Object Architecture (BOA). An architecture is a set of rules to define the structure of a system and the interrelationships between its parts. The components within an architecture are the basic building blocks and tools. An architecture also contains patterns, which advise on how to combine basic components using the tools. The functions of a BOA are to represent the components that are used to 'model' the business problems and build the system [2].

3.3 BOA Framework

As with Business Object, there is no standard way to develop a BOA. Our approach is to adopt both top down and bottom up directions as shown in Figure 3. After defining the business process from a high level, we then identify all the necessary entity objects from the bottom up. Finally, we develop the business objects in the middle layer by collecting the appropriate business processes, functionalities, attributes and operations together. Those business objects do not hold the business processes or the entity objects. Instead, they point to them when they want to use them. Different business objects thus share a single business process or entity object. The benefit of it is that when we change any business processes or entity objects, the updated versions will point to the relevant business objects. Another benefit of the BOA is that not only can we reuse the business processes and the entity objects but also we can reuse the business objects as a package. The reuse of business object will substantially improve software quality as developers can reuse the pre-defined and pre-tested object components.

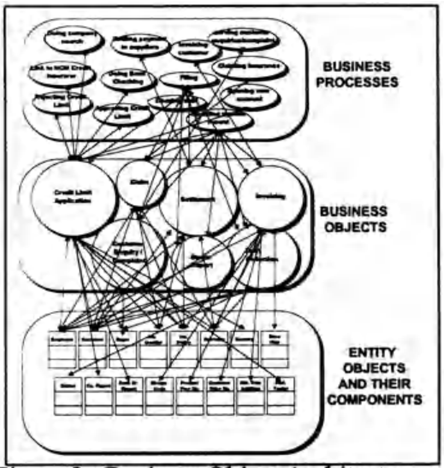


Figure 3 : Business Object Architecture

4. Business Object Repository / Reuse Library

4.1 Business Object Repository

The BOA framework emphasises the reuse of business processes and object components, the aim of the business object repository is to materialise it. Figure 4 shows the infrastructure of a business object repository in which the entity objects are at the central layer representing the common database in an organisation. The business processes, which normally form the common functionality in the business domain, are situated in the second layer. The outside layer contains different business objects. Both the entity objects and business processes are shared by the business objects.

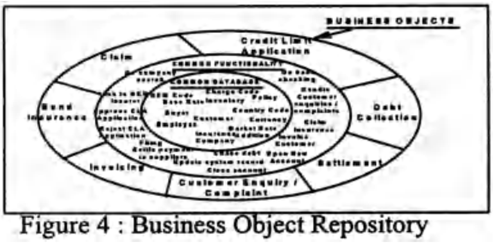


Figure 4 : Business Object Repository

4.2 Reuse Library

Ideally a reuse library is to be run in a CASE tool environment. As illustrated in Figure 5, the business process and entity object directories are the sub-directories of the business object directory. For example, if we click open the credit limit application (CLA) business object folder, the sub-folders of business process and entity object will prompt out on the screen. If we click open these sub-folders, all the relevant business processes and entity objects files will display on the screen. In this business object directory, those business processes and entity objects are "read-only" files. If we want to edit any of these files, we need to go to the business processes and entity objects main directories to do the changes. The updated versions of the business processes and entity objects will point back to the sub-directories of business processes and entity objects in business object main directory. The benefit of which is that we only have to change once no matter how many times the business processes and entity objects are reused by different business objects.

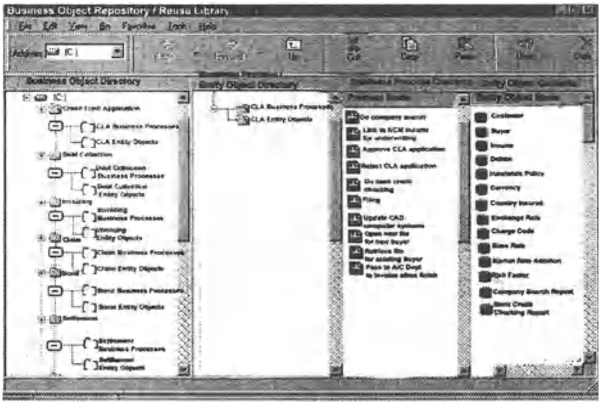


Figure 5: Reuse Library

5. Dynamic Systems Development Method

The Dynamic Systems Development Method (DSDM) is a formalisation of Rapid Application Development. Other than rapid development, DSDM also forms a vehicle to drive the IT developers and the business end-users together through its holistic approach such as emphasis on substantial end-user involvement, joint application development and joint requirement planning, function points, time-boxing, clean room technique, project estimation, usability testing, configuration management, change control, quality assurance and software procurement.

Traditionally, developers tend to put a subjective view on their work presuming that is what the real world needs. DSDM's fundamental assumption is that nothing is built perfectly first time. As a result all steps can be revisited as part of its iterative prototyping life-cycle. Therefore the current step needs be completed only enough to move to the next step. DSDM not only provides a life-cycle but also the necessary controls to ensure its success.

6. Dynamic Business Object Architecture

6.1 The Framework

The DBOA framework showing in Figure 6 is aimed at throwing one stone at two birds namely the 'quality' bird and the 'time & budget' bird. It is an integration between the BOA and DSDM. Amongst each life-cycle there is an incremental prototyping approach through these phases moving anti-clockwise from the top with feasibility studies, functional prototype, design prototype and implementation. The black arrows show the transfer points from one phase to the next and the grey arrows show where the development can easily return to an earlier phase. The white arrows indicate that the BOA model can always be re-architectured at different stages of the project phases.

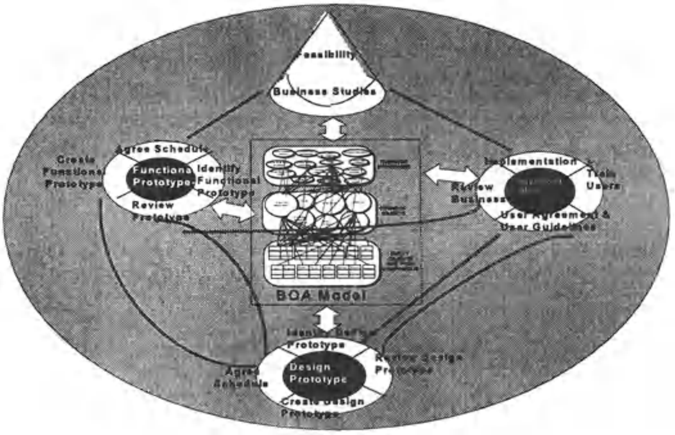


Figure 6 : Dynamic Business Object Architecture

6.2 Case Study Implementation : A Major Debtor Profile System for a Credit Insurance Agent

6.2.1 Background of the Project

The case study was carried out at a credit insurance agent, CAD Consultants Ltd.(CAD). When there is a transaction between seller and buyer, the buyer is given a certain length of credit period after the receipt of goods. The seller then insures the value of the products. Credit insurance is to protect the sellers (i.e. the insurance policy holders) from insolvency if their buyers fail to pay after the credit period. It is a commercial coverage by a contract binding a party to indemnify another against specified trading loss in return for premiums paid. CAD not only manages insurance policies on behalf of its customers but also has to determine the 'Credit Risk' of each buyer as well as the global risk exposure (political, economic, geographical) of the buyers' countries. This case study is to develop a Major Debtor Profile System to monitor the debt exposure. Under the credit insurance terms and conditions, any buyers who have credit are referred as debtors. The purpose of this project is to provide decision support to the business end-users on the approval/rejection of any future credit insurance applications.

6.2.2 Joint Requirements Planning (JRP) and Joint Application Development (JAD)

A 'Project Development Team' was formed consisting of three IT developers and four business end-users. After the initial JRP meeting of feasibility and business studies, JAD meetings took place at the end of each of the time-boxes. These took the form of a review of that time-box phase, and identifications of requirements for the next phase. During these 'End-of-project-phase' meetings, end-users were invited to test the prototype on-screen. The IT department staff would collect the end-users' comments and feedback to modify the prototype.

6.2.3 Project estimation by using function points

The project started with an estimation of the project size by using function points which are a method of estimating the "amount of functionality" required from an application and is also used to estimate project completion time and resources (human and finance) required. The basic idea involves counting screen inputs and other features of a description of functionality [8]. Figure 7 shows the essentials of this estimation.

The estimation works by identifying each function as easy, medium or difficult in terms of expected development 'complexity'. Function points were set after the Joint Requirements Planning (JRP) meeting with the end-users where requirements were obtained from them. As function points are the units used to measure the project, should there be changes in the user requirements during the project phases, the function points table will need to be re-scheduled accordingly. It is important to note that within a fixed timescale, it would be impossible to accommodate extra functionality without changes to the function point estimation. Therefore there are significant implications for the cost and/or duration of the project if such changes are required.

Functions	**Points Allocated (1=easy; 2=medium; 3=difficult)**	**Estimated developer-hours (easy=6 hours; medium=12hours; difficult=18hours)**
Declared Month / Year (user entry)	1	6
Customer No. (user entry)	1	6
Customer Name (auto display)	1	6
Policy No. (auto display)	1	6
Policy Name (auto display)	1	6
Client Reference No (user entry)	1	6
CAD Buyer No. (auto display)	1	6
Buyer Name (auto display)	1	6
Country Code (auto display)	1	6
Amount (user entry)	1	6
Currency Code (auto display)	1	6
GBP Equivalent (auto display)	1	6
Insured Limit (auto display)	1	6
Total amount of debt (in GBP equivalent) for a particular debtor should be added up and shown at the bottom of the screen (auto display)	3	18
The entry of Major Debtor Profile record is done monthly. Transfer previous month to history	2	12
The new screen for entry of current month's record will be cloned from previous month and the end-users overwrite it.	1	6
Report by Debtors	1	6
Report by Currencies	1	6
Report by Countries	1	6
Report by Customers	1	6
Report by Period (i.e. monthly record)	1	6
Report by Amount (in GBP)	1	6
Report by Teams	1	6
Update Main Menu	1	6
Total :	27 points	162 hours

Figure 7 : Function Point for Major Debtor Profile System

6.2.4 Time-Boxing

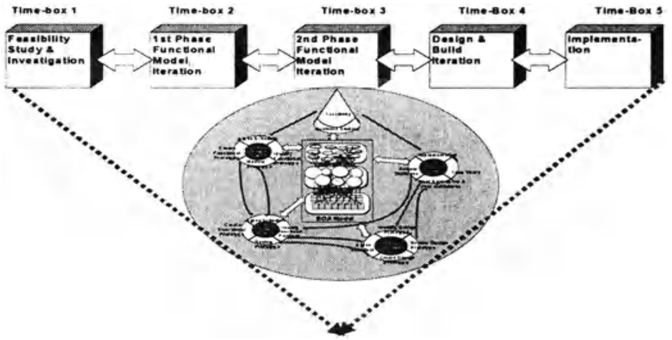

Time-box 1
Time-box 2
Time-box 3
Time-Box 4
Time-Box 5
Feasibility Study & Investigation
1st Phase Functional Model Iteration
2nd Phase Functional Model Iteration
Design & Build Iteration
Implementation
BOA Model

Figure 8 : Time-boxing for Major Debtor Profile System

As suggested by the DSDM manual, "Requirements can change, time can never slip" so heavy emphasis is placed on the importance of time-boxing technique as shown in Figure 8 to ensure project will be delivered on time thus within budget. DSDM defines time-boxing as 'setting a deadline by which a business objective must be met', and suggests that the boxes are set for the clearly defined delivery objects. This project consists of five time boxes namely:-

- ***Time-box 1*** **: Feasibility / Business Studies :** To get to know the user requirements. The project provided quite clear-cut requirements, in the technical context of a general need to provide information on all the current debtors for a particular customer. The Major Debtor Profile Interface is in fact a consolidation of different database files such as Customer, buyer, Credit Insurance Policy, Country, Currency and Exchange Rates. The Major Debtor Profile system correlates relevant data fields from different database files to the Major Debtor Profile Interface. When the users enter the customer reference number, this customer reference number data field will trigger other correlative data fields to display all the existing debtors' details for that customer as well as the insured limit where CAD's customer is allocated to each buyer. Over and above these features, the end-users can also view a profile by currencies, countries and period etc. This Major Debtor Profile system is to assist the end-users in making decision whether to approve or reject any future credit insurance application on a particular buyer.
- ***Time-box 2*** **: 1st Phase Functional Model Iteration :** To produce a version of the working system, from analysis and design model, notation to implementation, that could demonstrate to the user the essential features required to enable a user view the Major Debtor Profile and to test the prototype.
- ***Time-box 3*** **: 2nd Phase Functional Model Iteration :** To provide essential functionality to the model. The first phase had been mainly concerned with user interface design, with little in the way of 'business functionality'. An important issue of this early work in Phase 2 was to revisit the BOA framework

and its break-down to find out whether the conceptual model was the right shape to drive through to deliver systems / applications to the business.

- *Time-box 4* : **Design and Build Iteration** : To design and actually build the system. Hence by the end of this phase the system must contain absolutely all functionality, in a form which is suitable for testing.
- *Time-box 5* : **Implementation** : To test the system with the end-users which was purely to test the reliability and to debug the system. No extra requirements from the users were accepted within this Time-box. If the user had wanted to make further changes, we would have had to reschedule the project development life-cycle. During this project, end-users' approval was obtains and User Guidelines were prepared. Staff training programmes were conducted before system configuration, data take-on and system went live.

6.2.5 Prototyping Strategy

As time-boxing controls the pace of design and development that is so essential to the project, computer based techniques make this feasible. It would indeed be quite impossible to entertain the idea of tight schedule time-boxes without a means of maintaining both design documentation and implementation in a flexible and responsive way. The implication for the quality of the delivered product is quite clear. Therefore, the criteria of both the Business Object design model and interface prototype must clearly reflect the business, be flexible to change, quick to build / assemble and support reuse. In this project, a prototype strategy was to produce a version of the working system. During the feasibility and business study phase, a major debtor profile business object was created as shown in Figure 9.

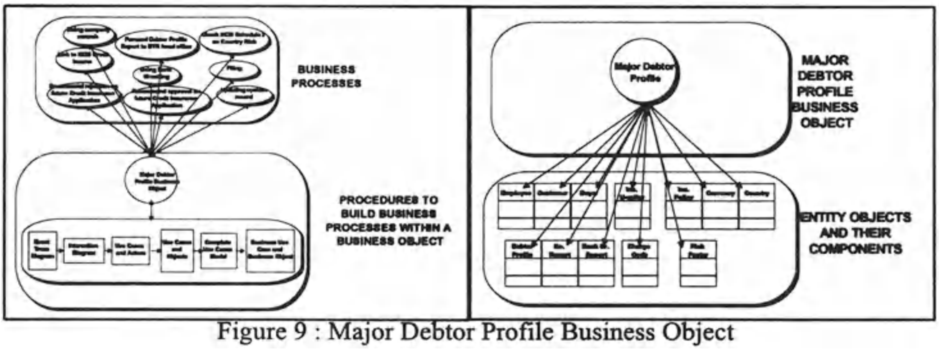

BUSINESS PROCESSES
PROCEDURES TO BUILD BUSINESS PROCESSES WITHIN A BUSINESS OBJECT
Major Debtor Profile
MAJOR DEBTOR PROFILE BUSINESS OBJECT
ENTITY OBJECTS AND THEIR COMPONENTS

Figure 9 : Major Debtor Profile Business Object

Figure 9 is the breakdown of Figure 3 (Business Object Architecture) specifying one single aspect of the Major Debtor Profile. On the left hand side of Figure 9, a business process model starts with an Event diagram followed by an Interaction diagram, Use Cases and Actors, Use Cases and Objects, Complete Use Cases Model and Business Use Cases and a Business Object diagram. On the right hand side of Figure 9, relevant Entity Objects and their other components are identified to be used for the Major Debtor Profile Business Object.

The Unified Modelling Language (UML) notation [17,4] was chosen as the design notation for this project, as shown in Figure 10.

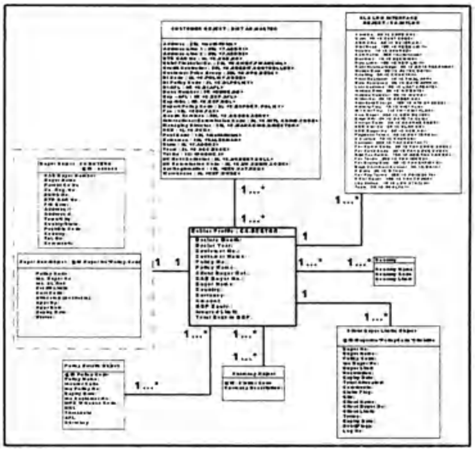
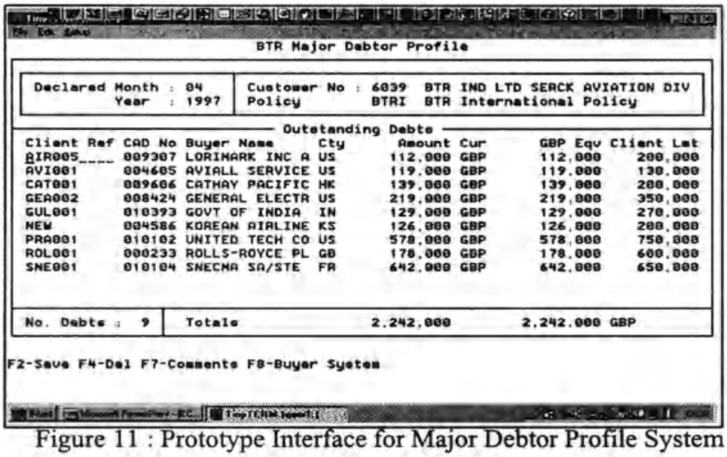


Figure 10 : UML notation

The interface was constructed using System Builder Plus [4] 4GL GUI tools to develop the interfaces that could be demonstrated to the users the essential features required to enable the application to be developed as shown in Figure 11.



```
                    BTR Major Debtor Profile

 Declared Month : 04    Customer No : 6039  BTR IND LTD SERCK AVIATION DIV
          Year  : 1997  Policy        BTRI  BTR International Policy

 ---------------------- Outstanding Debts ------------------------------
 Client Ref CAD No Buyer Name      Cty      Amount Cur      GBP Equ Client Lmt
 AIR005____ 009307 LORIMARK INC A  US      112,000 GBP      112,000    200,000
 AVI001     004605 AVIALL SERVICE  US      119,000 GBP      119,000    130,000
 CAT001     009606 CATHAY PACIFIC  HK      139,000 GBP      139,000    200,000
 GEA002     008424 GENERAL ELECTR  US      219,000 GBP      219,000    350,000
 GUL001     010393 GOVT OF INDIA   IN      129,000 GBP      129,000    270,000
 NEW        004586 KOREAN AIRLINE  KS      126,000 GBP      126,000    200,000
 PRA001     010102 UNITED TECH CO  US      578,000 GBP      578,000    750,000
 ROL001     000233 ROLLS-ROYCE PL  GB      178,000 GBP      178,000    600,000
 SNE001     010104 SNECMA SA/STE   FR      642,000 GBP      642,000    650,000

 No. Debts :  9 | Totals            2,242,000        2,242,000 GBP

F2-Save F4-Del F7-Comments F8-Buyer System
```

Figure 11 : Prototype Interface for Major Debtor Profile System

7. Project Evaluation

The 'Major Debtor Profile System' project had been delivered on time and was in operation. The BOA model was considered to be satisfactory as a vehicle to communicate with the end-users and interpret the business requirements for the new system and how it linked with other business objects. At the end of the project, the user community was satisfied, and had clearly felt very much involved in the whole process. The final success of the project was felt by the complete 'team'; not only the developers, but also the equally essential end-users who had been so actively involved in the JRP and JAD sessions and the user acceptance testing. The result of this case study has highlighted a few areas in which we feel that the DBOA showed particular strength. These areas have also painted a picture of a successful interpretation of the method for a DBOA development in that:

- The reuse of existing business processes and entity objects within the BOA has reduced the development time and effort.
- The gap between the conceptual model and software implementation had obviously been narrowed. Had it not been the DSDM approach, we would not be able to check whether our conceptual BOA model is the right model for the business. The definition of good quality is largely the suitability to the business.
- Communication between developers and end-users was much better. The users were very much involved, to the point where 'the team' was quite definitely a description applicable to the mix of people, developer and user, involved in the project. There was an integration of the two roles; a change of relationship from supplier/consumer to partnership. The final system was our system, not their system. Equally significantly, if not more so, the users enjoyed the experience of taking responsibility for their own system. It is also worth mentioning that the experience was (most of the time!) enjoyable for the developers.
- The holistic approach, as a result of this partnership, has enabled the developers to obtain a better understanding of the business and its requirements. The intensity and effectiveness of the JRP and JAD sessions was beyond any doubt. The concept of getting the right people to concentrate exclusively on the problem, and of empowering them to make the right decisions, paid off. And because of the heavy involvement of the business end-users, an IT project has become more of a business project. This is consistent with the prototyping that the function of IT support is to solve business problems.
- The iterative approach to design worked. It enabled us to revisit the BOA conceptual model and modify it in response to the changes in circumstances. The first functional prototype was very much imperfect. But at least it was something for the user to work with. The process of refinement which went on through Time-boxes 2, 3 and 4 resulted in numerous opportunities to fix the imperfections.
- We met, with comparative ease, what would have been an impossible deadline using the conventional life-cycle.

The whole rationale of this paper is "To achieve the objective of delivering quality software on time and within budget". With this two-way echo between the developers and the end-users, we consider we have successfully brought these acronyms (BOA and DSDM) together through our experience obtained from the above case study. Such synergy quickly and effectively reacts to the business changes.

8. "S.M.A.R.T." Evaluation Criteria

An evaluation criteria framework called "S.M.A.R.T.", based on the characteristics of both the BOA and DSDM technique, has been developed to evaluate the DBOA schema in terms of:

'*S*'calable: As each business object component is individual, we can always increase the number of the business object without affecting the integrity of the existing one.
'*M*'easurable: Function Points were used to measure the size and complexity of the system. User Acceptance Testing was also used to measure the satisfactory level of the end-users on the system.
'*A*'chiveable: The holistic approach of DSDM life-cycle environment has increased the interactions between the end-users and the developers. Communication between them has thus been improved to enable the IT developers to deliver a software more achievable to the business requirements.
'*R*'eusable: The sharing of entity objects and process objects amongst business objects is the classic way of object reuse. Business objects themselves can also be reused as a package as well.
'*T*'ime-manageable: The time-boxing technique has provided a good control of time management to run project in order to deliver the system on time and within budget.

9 Conclusion And Further Research

9.1 Conclusion

In this paper, we present a DBOA to recommend a strategy for managing the time and budget against quality issue. An implementation of this is also presented through an insurance project case study. The DBOA techniques used in this paper should also be applicable to projects in any other business sectors. Although the result of the above case study is considered to be successful, DSDM is still not a mature technology. There are several 'challenging' areas where we would have to warn the developers when using the DBOA approach:-

- ***Friction between developers and end-users*** : there is always a situation where the developers and the end-users do not get along well.
- ***How to select the "right" people and to empower them to make "right" decisions?***: this is more to do with business issues and it can only be improved through experience.
- ***Time-boxing Syndrome***: everything is set inside a time-scale agreed with the business end-users. If planning is insufficient, developers would juggle between time-boxes. They will be forced to omit some unfinished tasks if they overrun the time-boxes or get panic to catch up at later time-boxes or they might have to abandon project if under pressure.
- ***Work Pattern / Paradigm shift for developers*** : the boundary between IT and business world is taken away. Developers have to cross the border to communicate with the business end-users and to experience business environment rather than developing the system in their own environment.

9.2 Further Research

Currently, we are investigating how to tackle the problems arisen from the conflict between developers and the end-users as well as the time-boxing syndrome. In the meantime, a business object repository is under construction using the Rational Rose

Version 4.0 UML CASE tools. Furthermore, a multiple projects case study will be carried out using the DBOA model to deal with complexity management.

10 References

1 Casanave C, Standardised Business Objects, Conference of Building & Using Financial Business Objects, London, UK, 22-23 October 1997 (http://www.omg.org).

2 Casanave, C, Business Object Architectures And Standards, Proceedings of Object-Oriented Programming, Systems, Languages & Applications (OOPSLA)'95 Conference Business Object Design And Implementation Workshop, Austin, Texas, USA, October 1995 (Eds. Sutherland et al) (Springer-Verlag, London) pp 7-28.

3 DSDM CONSORTIUM, Dynamic Systems Development Method Manual Version 3.0, 1997 (Tesseract Publishing, Surrey, UK).

4 Fowler M, UML Distilled - Applying The Standard Object Modelling Language, 1997 (Addison-Wesley Longman, Massachusetts).

5 Graham I, Migration to Object Technology, 1994 (Addison-Wesley, N.Y).

6 Hartha W et al, An Architecture Framework: From Business Strategies To Implementation, Proceedings of Object-Oriented Programming, Systems, Languages & Applications (OOPSLA)'95 Conference Business Object Design And Implementation Workshop, Austin, Texas, USA, October 1995 (eds. Sutherland et al) (Springer-Verlag, London) pp 47-60.

7 Hung K et al, A Dynamic Business Object Architecture For An Insurance Industry, Proceedings of Object-Oriented Information Systems (OOIS)'97 Conference, Brisbane, Queensland, Australia, 10-12 November 1997 (Eds. Orlowska et al) (Springer-Verlag, London) pp 145-156.

8 IFPUG, International Function Point Users Group, Ohio, USA (http://cuiwww.unige.ch/OSG/FAQ/SE/se-faq-S-2.html.#S-2).

9 Jacobson I et al, The Object Advantages: Business Process Reengineering With Object Technology, 1994 (Addison-Wesley, New York).

10 Jacobson I, Use Case Engineering Tutorial, Object-Oriented Programming, Systems, Languages & Applications (OOPSLA)'96 Conference, San Jose, California, USA, October 1996.

11 Martin J, Rapid Application Development, 1991 (Macmillan, New York).

12 OMG, Object Management Group - Object Management Architecture Guide, 1995 (John Wiley & Sons, Inc., New York).

13 Ramackers G et al, Object Business Modelling Request & Approach, Proceedings of Object-Oriented Programming, Systems, Languages & Applications (OOPSLA)'95 Conference, Austin, Texas, USA, October 1995 (Eds. Sutherland et al) (Springer-Verlag, London) pp 77-86.

14 SB+, System Builder Plus Version 2.3 Developer and Administrator Guide, 1995 (System Builder Technology UK Ltd., Prestbury, UK).

15 Stapleton J, DSDM: The Method In Practice, 1997 (Addison-Wesley, Essex, UK).

16 Sutherland J, The Object Technology Architecture: Business Objects For Corporate Information Systems, The 1995 Symposium for VMARK Users, Albuquerque, USA, 1995 (http://www.tiac.net/users/jsuth/).

17 UML97, Unified Modelling Language Version 1.0, January 1997 (http://www.rational.com).

18 Young J et al, Time For IS Professional To Come Out Of The Closet And Join The Party - Using A Strategic Change Framework To Understand The Influences on IS, 7th Annual Business Information Technology (BIT)'97 Conference, Manchester, UK, 5-6 November 1997 (Manchester Metropolitan University CD-ROM).

OBJECT-ORIENTED SOFTWARE DEVELOPMENT WITH REUSE

Milankovic-Atkinson M. & Georgiadou E.
University of North London
School of Informatics and Multimedia Technology
2-16 Eden Grove, London N7 8EA
tel: +44 (0) 171 753 5127, +44 (0) 171 607 2789 fax: +44 (0) 171 753 7009
email: m.atkinson@unl.ac.uk, e.georgiadou@unl.ac.uk

ABSTRACT

Improved productivity in Object-Oriented technology results from the reuse of software. development with reuse in mind brings design and implementation closer. This requires a software tool to support both top-down development, from the specification of classes as icons in the graphical representation of the Object Model [1] to code generation, and bottom -up development, from components in the source code to their class representations as icons of the Object Model. To make this possible the tools needs to be fully integrated into the development environment.

This paper discusses facilities offered by a number of currently available tools and our experience of using the CASE tool Together/C++ [2] and animator Look![3] as well as the Borland C++ 4.5 for Windows development environment [4].

Initially we provide a brief discussion of different approaches for software reuse such as MOOSE [5], development through formal specification (VDM) [6], and Eiffel [7]. We report of lessons learned through the reuse of existing software components in C++ from generally available proprietary public domain or shareware.

We discuss issues of understandability and complexity particular to object-oriented systems and the usefulness of animators such as Look!. Static analysis of object-oriented code cannot deal with ambiguities caused by polymorphic routine calls e.g. tracing remote functional dependencies [8]. Animators can facilitate understanding of the additional complexities.

With the improved understanding of object-oriented metrics [9,10,11], and their usefulness in predicting effort and cost of software reuse [12] we propose that a selection of design and performance metrics could be collected and maintained as part of the software components [13, 14] as an integral part of the development process towards continuous quality improvement through reuse.

C. Hawkins et al. (eds.), *Software Quality Management VI*

INTRODUCTION

Software development based on the reuse of standard components is not a new idea. In 1968 Doug McIllroy at the first NATO Software Engineering Conference [15] stated that the software industry lacks a software component subsidiary which would enable mass production of software. As in all mass production industries both quality and productivity would improve because software would be constructed using reliable components, and new applications using these components could be developed more rapidly.

Object-oriented languages made it technically possible to produce and package software in component form and enable reuse on this scale. The unit of reuse at the lowest level of granularity is a class, an implementation of an ADT offering better encapsulation as well as facilities for extending and customising. If used as a 'black box' component, its services are exploited through official interfaces while the implementation can remain hidden. If used to be modified through inheritance, a new version of the class can be derived using the existing class as a base, the additional attributes and /or services may be added, while those provided by the base class reused or modified. Multiple inheritance allows characteristics of several different classes to be combined to produce a new class.

Software development with reuse starts with the high level design by specifying the system architecture. These specifications are then used to try and find suitable reusable components. In some cases the existing reusable components form the basis around which the new system will be developed. This makes the development process reuse driven. The design is based on the available reusable components. Even the requirements may be modified to accommodate the use of these components. Software development with reuse, therefore, differs from the 'one off' development because it brings design closer to implementation and vice versa.

DIFFERENT APPROACHES

In order to facilitate reuse it is necessary to have a means for enabling the selection and evaluation of the components being considered for reuse, and an environment that will support the integration of the selected components into the new system by providing facilities for both 'top down' and 'bottom up' development.

One approach for managing this type of development combine the software development for reuse with the software development with reuse [16] by means of a dedicated development environment - CASE tool/Method repository/ language, in which the reusable components are designed and implemented and new applications generated. The reusable software is therefore designed and implemented using the built in standards specific to the development environment.

MOOSE (Method for Object-Oriented Software Engineering) developed at the University of Sunderland [5] covers the entire lifecycle in which reuse represents a specific stage. All the deliverables of the development process from specification analysis/design diagrams to source code are stored in a persistent CASE tool repository all of which can be reused. The software components developed using the MOOSE tools are specified using formal specification techniques which provide precise descriptions of the reusable components. The Distributed Environment for Engineering Reuse (DEER) enables the repository components created using the MOOSE method and tools, the deliverables of the software development for reuse, to be viewed and manipulated e.g. loaded into the MOOSE tools for the software development with reuse. DEER provides a 'browsable hypertext/hyperCASE view' of the repository and is accessible through the www using an extended version of HTML for selecting the components which will be displayed in MOOSE notation offering a variety of views. The idea is attractive, the restriction is that in order to be able to reuse components and access them through DEER the components must previously have been produced using MOOSE, which means investment in the development of a component library, and development of new software using the MOOSE/DEER software tools.

The VDM based software Component Retrieval VCR [6] has a more formal approach. The programmer should be able to locate software components that will exactly match their needs, to overcome possible errors being introduced when reusing software components, making them less reliable. The software components are developed and implemented so as to carry exact semantic information about themselves which will be then used for deduction-based component retrieval. Each component is characterised by its VDM specification in the form of pre and post conditions and type signature. The type signature can be computed using type inference techniques. The pre-conditions specify the conditions necessary for a function to be called, the post-conditions express the guaranteed conditions after the the function has been executed. This information is used for component retrieval. The search keys also consist of the type signature, pre- and post- condition triple. A component will match if it has a provable compatible signature and specification. The specification matching is performed as a sequence of successive filtering steps A component specification will be compatible with the search key if it simultaneously has a weaker pre-condition and stronger post-condition. The emphasis is on high precision retrieval.

Eiffel is a language and environment designed so as to maximise the benefits offered by the object-oriented technology and to enable the 7]. The class definitions contain assertions that define the most important semantic properties of the class and its services. These are formally defined for each service (class member function), pre-conditions are introduced with a keyword *require*, and post-conditions with the keyword *return*. The class *invariants* must be satisfied at all times, whenever objects of a class change state. An assertion violation can throw an exception. The preconditions, postconditions and invariants implement programming by contract. The rights and duties for every component are thus precisely indicated making their reuse easier. The graphical Eiffel browser GOOD makes it possible to retrieve classes from the class libraries [17].

This approach is neat, elegant and safe, but reduces the amount of potentially available software that could be considered for reuse, and requires the investment in the development of a library of software components as well as the software development environment. It would be desirable to find a way of reusing the generally available software.

Our approach was to concentrate on the reuse of generally available and proprietary software choosing a suitable design method supported by a software tool for analysis/design/implementation that would not require unwarranted commitments and would be compatible with our existing software development environment. Additional important requirements were seamless transition from design to implementation and support for maintaining the design documentation up to date with the code at all times in machine processable form, so that changes in the development would not require additional documentation effort. We have used the latest version of the Coad method [1], Together/C++ version 1.2 [2] were used with Borland C++ 4.5, Windows 3.1[4] running on a Novel Netware 3.12 with 100+ PC 386 and 486 workstations on three file servers.

THE COAD METHOD AND SOFTWARE DEVELOPMENT WITH REUSE

The Coad method uses the same Object Model, shown on Figure 1 below, from

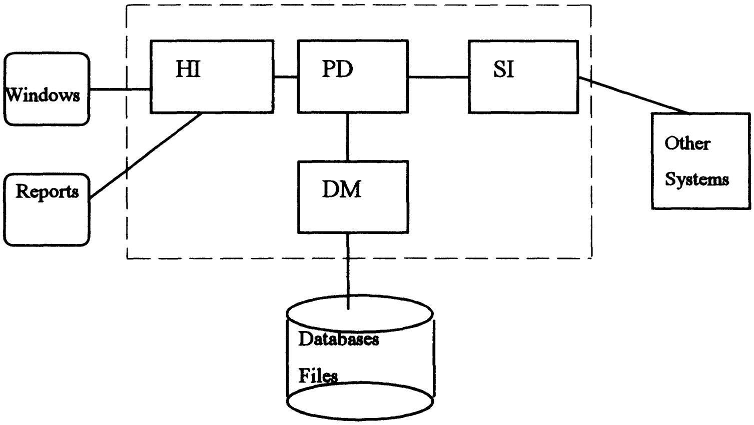


Figure 1 -Components of the Object Model

initial analysis to implementation, continuously refining it until fully implemented. The Object Model representing the new object-oriented application consists of distinct component types. The Problem Domain component (PD) will model the objects/classes most closely related to the problem at hand. The Human Interaction component (HI) will deal with the system interactions: the requests for handling the events, and the responses and would normally include the Graphical User Interface objects, reports, etc. The Data Management component (DM) will handle the data management complexities. This will include the data structure objects used during program execution, and objects for handling persistent data that will provide file handling services for interacting with physical files and databases acting as an 'object server'. The System Interface component (SI) will handle the interactions which the currently developed system with other systems or devices.

This component partition encourages the development of applications where the Data Management component and Human Interaction component will be largely created reusing available classes from the template class libraries of the development environment.

The Borland C++ Development environment [4] organises these classes into container classes called Borland International Data Structures (BIDS) that provide data structure objects such as linked lists and dynamic arrays and include features to support archiving objects in files, and Object Windows Library classes (OWL) for building Microsoft Windows applications, including classes that simplify the use of the Graphics Device Interface (GDI).

REUSING THE BORLAND INTERNATIONAL DATA STRUCTURES LIBRARY CLASSES

BIDS is a library of class templates organised into two categories namely: Fundamental Data Structures (FDS), and Abstract Data Types (ADT). BIDS implements the data structures according to their standard behaviour. The 4.5 version defines three FDS container types: Vectors, Singly and Doubly-linked lists (Table 1). The ADT comprises eight container types: Array, Sorted array, Stack, Queue, Deque, Bag, Set and Dictionary. ADTs are implemented using one of the FDS containers. For each type of FDS, or ADT data structure there are many differenct containters. The name of each container class has coded prefixes, after the letter T for template, indicating the nature of the class. M indicates that memory allocation is not standard but separately specified. C means counted, S sorted , I indirect, CI counted indirect, SI sorted indirect etc. Sorted containers require sortable objects, therefore the relevant relational operators need to be overloaded for the class. Objects of any type can be stored and manipulated by a container: predefined scalars, or objects of previously defined classes. These objects can be stored: directly or indirectly Direct storage means that the object is copied into the storage allocated to it, producing a physically independent copy, of the original . Therefore if direct storage is used objects must have a copy operator defined. As a consequence direct storage containers can only handle homogeneous objects. With indirect storage, a pointer to an item is stored in the container, so objects of any type including heterogeneous objects can be held as long as they are derived from a common ancestor.

Table 1 BIDS template classes for doubly linked lists:

Class Template Name	Meaning
`TMDoubleListImp`	Double-linked list with managed memory allocation
`TDoubleListImp`	Double-linked list using standard memory allocation
`TMSDoubleListImp`	Sorted double-linked list with managed memory allocation
`TSDoubleListImp`	Sorted double-linked list using standard memory allocation
`TMInternationalIDoubleListImp`	Double-linked list of pointers with managed memory allocation implemented though `TMDoubleListImp`
`TMIDoubleListImp`	Double-linked list of pointers with managed memory allocation implemented through `TMInternaionalIDoubleListImp`
`TIDoubleListImp`	Double-linked list of pointers using standard memory allocation
`TMISDoubleListImp`	Sorted double-linked list of pointers with managed memory allocation

In order to create and use a container it is only necessary to define the required class using the template, by specifying the type of object it will hold, and familarise oneself with the interface (service names, parameters and return types) See Table 2. For example the following is a declaration of a double-linked list called *DataList* of objects of class *TheData* using the `TDoubleListImp` template :

```
TDoubleListImp <TheData> DataList;

//class TheData should have a default constructor and

//meaningful       copy semantics.
```

The FDS and ADT containers also provide iterator classes to help access the content of a container. The subscript operator [] is to be adequate for vectors and arrays since they have the notion of an index while other container types such as lists, stacks, bags etc. do not, and the iterators simplify access to the objects they handle. The double-linked list has the following iterators:

```
TSDoubleListIteratorImp       TMIDoubleListIteratorImp
TMSDoubleListIteratorImp      TIDoubleListIteratorImp
TMISDoubleListIteratorImp     TISDoubleListIteratorImp
```

Table 2 Services offered by the double linked-list containers

Service	Description
`int Add(const T& t)`	inserts an object at the head of the list. `t` is the reference to the inserted object
`int AddAtHead(const & t)`	inserts an object at the head of the list. `t` is the reference to the inserted object
`int AddAtTail(const T & t)`	inserts an object at the tail of the list. `t` is the reference to the inserted object
`const T & PeekHead() const`	returns the reference to the head list element
`const T & PeekTail() const`	returns the reference to the tail list element
`int Detach(const T& t,int del=0)`	removes the first occurrence of a specified object from the list. `t` specifies the object to be removed. The parameter del specifies whether to destroy the object after removing it from the list
`int IsEmpty() const`	returns 1 if the list object is empty, and 0 if it is not.
`int GetItemsInContainer() const`	Returns the count of number of objects in the container
`void Flush(int deleteMode = 0)`	deletes all list elements without removing the list structure

Reusing code by defining specific classes from the available templates reduces development time and risk, increases reliability, introduces and encourages standardisation, but an even greater benefit comes from the flexibility that it provides. Examples of this are performance tuning of a completed application caused by changes in the requirements. For example the application may need to handle a larger number of insertions and deletions than previously expected, the number of objects manipulated may increase greatly, or faster response times may be necessary etc. Substituting the underlying data structure or the particular implementation of the chosen data structure can often help successfully alleviate the problem To change the container class would require very few adjustments to the code. This modification may take place at any stage during development or maintenance. It would also be possible to empirically test the performance of the application by trying out different container class templates using the profiler in order to select the most suitable one.

TOGETHER/C++

Figure 2 represents the Together/C++ development environment shown inside the dotted line. A solution to a problem in object-oriented development with reuse is

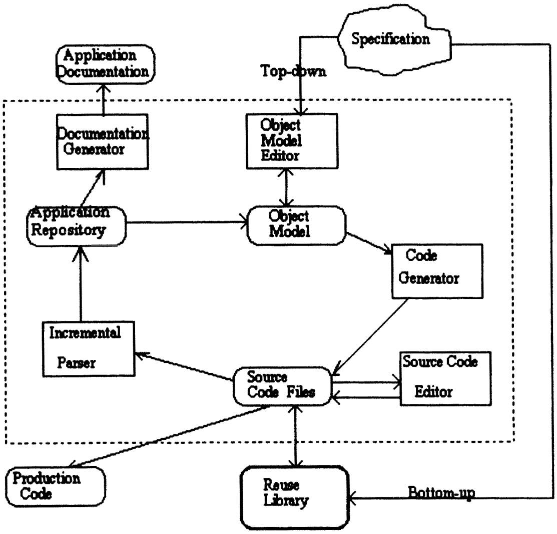


Figure 2 The Development Environment

derived by determining the object classes, their responsibilities and structure from the requirements specification which is 'top down'. The object model editor can be used for creating the object model diagram with icons representing the classes of the system (i.e. the object layer), the class attributes (attribute layer), the class services,(service layer) the relationship among classes (the structure layer), and the system can be partitioned into subjects (the subject layer). A subject represents a collection of classes. While creating the object model diagram using the object model editor - the code in the form of class skeletons representing the icons, service prototypes representing the service names, and attribute definitions is generated concurrently. The text editor can be used to add more detail i.e. write the code for the services etc. and these changes, if correct, will simultaneously update the object model diagram.

Including the available reusable components in the new system is 'bottom up' development. The implemented component being reused is imported into the design. It is, therefore, necessary to be able to work in a software development environment which will support such development in other words, be capable not only of carrying the design through to the implementation but also to extract from the source code the design information and add it to the design of the system that is being developed. When C++ classes are added to the object model from source code, the relevant icons representing them in the object model diagram will be generated. This will include the inheritance relationships, as these are supported and directly implemented in the language, while the client/server relationship connections need to be added to the model, as there is no single direct mapping of these in the code. These logical relationships can be implemented in C++ in a number of different ways.

The source code generated using the object model editor and text editor will be in the form of header and .cpp files and can be used as production code for testing . The entire system is C++ sensitive, and is parsed before any part of the model is placed in the application repository, and adopted as part of the object model. If errors are detected, error messages will be flagged and only when corrected the changes will be accepted, and the object model updated.

The documentation can be generated automatically. In order to include the diagrams the generated documentation can be loaded into e.g. Word, to add a contents page, and possibly customise before it is printed.

The versatile view management is a very useful feature. It can be organised for any of the 'layers' in several ways, the most powerful one is by setting filters.

It is difficult to keep the design documentation in line with the code that is continuously changing without CASE tool support. Continuously updating the design so that it reflects all changes to the software, throughout the life of the system is almost impossible to enforce if it requires extra effort. Together/C++ alleviates this problem.

COMPLEXITY ISSUES

Reusing the BIDS classes is relatively straightforward. This is because data structures represent a narrow domain that is well understood. Objects were created from classes from generic containers implemented using templates. The classes were reused through a form of instantiation, more or less as a 'black box'. They were generally not re-engineered.

The OWL classes were reused mainly through inheritance which meant that classes were derived by adding new attributes, services or by overriding existing services as white box frameworks and therefore code reuse that was not trivial. It was necessary to understand the code produced by someone else to be able to customise it [8]. The use of the debugger, and code animator Look! [3] were very helpful for understanding and testing the code because it had many facilities for dynamic analysis, with graphical animation of the creation of the objects, and the sending of messages which could be executed a step at a time. This was especially useful for tracing the execution of dynamic polymorphic calls.

Reusable code is the result of finding a solution to a specific problem, and then recognising that it may be applied to other problems. Reusable abstractions are developed 'bottom up'. Having successfully developed a system, the design could then be improved for future maintenance and reuse. Development *with reuse* is followed by development *for reuse*. The complexity of the software could be reduced by restructuring the design. For example overridden methods may indicate that the inheritance hierarchy may be better changed, or an abstract class introduced. Lack of cohesion among services of a class , may indicate that the class should be better to split etc.[9] As object-oriented software metrics for determining the complexity of classes and object-oriented systems are becoming better established [10] they could be applied for quality control of the software.

CONCLUSION AND FUTURE WORK

Object-oriented software with reuse requires a different approach to development. The classical structured methods separated analysis/design from implementation while here the design and implementation need to be closely integrated. We need to promote both 'top down' and 'bottom up' development. With multimedia, communications and database facilities becoming an integral part of applications it is increasingly important to be able to design new software in terms of reusable existing components. In other words, follow a reuse driven design process. It is also essential, as the resulting systems are becoming increasingly complex, to be able to control the management of the design process and be able to design the Problem Domain Component, Data Management Component and Human Interaction Component so as to produce high quality applications that can be properly tested and maintained. The design tool must be integrated with the development environment to support such development. The Coad method and Together/C++ have been found helpful in this respect. The code animator and debugger, and profiler are useful tools for understanding the complexity of reusable software, and improving software performance. The described development environment makes it possible to further improve fully implemented applications enhancing quality and reusability. The software can be re engineered, generalised, and developed into frameworks for similar applications.

Object-oriented technology enables production of reusable software components that can be used as part of new software products in a similar way to electronic components. It would be extremely useful to make them as reliable as electronic components This could be achieved by designing and implementing in a way in which software components could be tested, using automation similar to electronic components. Designing for testability [14] requires the specification of all the tests that are needed as part of the software component. Metrics would provide a means for calculating the number of tests, as well as for specifying the tests. One way of providing for the automation is to keep the metrics information as part of the software component [13]. These metrics would be calculated from the classes themselves and stored as part of the class information.

Different types of metrics both static and dynamic could be collected [12]. The dynamic metrics for a component would be collected and accumulated from information gathered during its use: e.g. number of exceptions thrown. If a component is derived by means of inheritance, the metrics could be inherited as well. The complexity metrics for a whole application could be calculated using the metrics of the comprising components [11].

We are currently organising a project for the development of software that will enable the automation of the collection of the software metrics, to be stored as part of the class. The collected information would be useful both in the developing of new software as well as assessing the suitability of software considered for reuse. The expected deliverables include the enabling software for automatic collection and recording of metrics, an improved understanding of the meaning of the metrics, and the significance of the different metrics in the development of object-oriented software and the possibility of introducing automated testing of object-oriented software.

REFERENCES

[1] Coad, P., North D., Mayfield M., Object Models Strategies, Patterns & Applications, Yourdon Press 1995

[2] Object International Ltd. , Together/C++ Manual 1994

[3] Object UK Ltd, Look! manual 1993

[4] Borland C++ 4.5 for Windows

[5] Ferguson R. I., Parrington N. F., Dunne P., MOOSE: A Method Designed for Ease of Maintenance, IFIP 1995, Chapman and Hall pp 37-44.

[6] Fischer B., Kievernagel M. and Struckmann W., High Precision Retrieval for High Quality Software, 4th Software Quality Conference, Dundee 1995, Proceedings Volume 1 pp. 80-88.

[7] Meyer B. Object-oriented Software Construction, Prentice-Hall 1988

[8] Cant S. N., Henderson-Sellers, B., Jeffry D.R., Application of Cognitive Complexity Metrics to Object-oriented Programs, Journal of Object-oriented programming July/August 1994 pp 52-63.

[9] Chidamber S.R., Kemerer C.F., A Metrics Suite for Object-oriented Design IEEE Transactions on Software Engineering, Vol. 20, No 6, June 1994, pp 476-491

[10]Lewis, J.A. Quantified Object-oriented Development: Conflict and Resolution, 4th Software Quality Conference, Dundee 1995, Proceedings Volume 1 pp 220-229.

[11] Kolewe, R. Metrics in Object-oriented Design and Programming, Software Development, October 1993, pp 53-62.

[12] Li, W., Henry S. Object-oriented Metrics that Predict Maintainability, J. Systems Software 1993; 23:111-22.

[13] Barnes, G. M., Bradley, R. S. Inheriting Software metrics, Journal of Object-oriented Programming, November/December 1993, pp. 27-34.

[14] Binder R.V., Design for Testability in Object-oriented Systems, Communications of the ACM, September 1994/Vol. 37. No. 9, pp 87-101.

[15] McIlroy, M.D. Mass Produced Software Components, Software Engineering Concepts and Techniques, Eds Buxton J.M., Naur P., Randell B. Van Nostrand Reinhold, 1976, pp. 88-98

[16] Sommerville I., Software Engineering, Addison-Wesley, fourth edition, 1992, pp.312-320.

[17] Meyer B. lessons from the design of the Eiffel Libraries, Comms of the ACM, September 1990/Vol 33, No 9 pp. 69-88.

Section 4

Software Tools

Integration of strategic, tactical and technical OO measurement

Marjan Heričko, Ivan Rozman, Matjaž B. Jurič, Romana Vajde Horvat
University of Maribor
Faculty of Electrical Engineering and Computer Science
Smetanova 17, 2000 Maribor, Slovenia
e-mail: marjan.hericko@uni-mb.si
URL: http://lisa.uni-mb.si/

Abstract

For measurement to become cost effective and efficient it should be directed by business goals. Since our goal when introducing object technology is to build better software it is necessary to perform technical OO measurement and relate it not only to tactical but also to strategic goals. In the paper we present some concepts of an environment that might help us to integrate and provide compliance between different aspects of OO measurement.

1. Introduction

According to [1] software measurement can be modeled using a three tired approach. Strategic measurement is concerned with long term organization's goals, tactical measurement is concerned with individual project's goals whereas technical measurement is concerned with details about particular product or process. Obviously, measures lower at the framework are more technology depended and specific. The focus of this paper is on OO metrics and its integration with project-level and organization-level measurement.

During our assistance in SPI (Software Process Improvement) efforts of Slovenian software companies (and departments) it has turned out that it is not easy to provide compliance between management and technical goals regarding software and software process quality issues. Therefore it is useful to provide an environment that integrates various aspects of quality assurance activities. In the paper we propose an environment that integrates ideas and results of our work on development of the PROCESSUS Methodology (methodology for assessment and improvement of software process) [2], methodology and tool for software product evaluation [3], and activities of our Object Technology Center [4]. We will concentrate on OO metrics that are really significant and important because they have huge impact on long-term software quality attributes such as reusability,

C. Hawkins et al. (eds.), *Software Quality Management VI*

interoperability, maintainability, testability etc. Technical OO measures that indicate if abstraction hierarchies (inheritance trees) are build, if encapsulation is violated and if attention is paid on providing weak coupling and strong cohesion of object systems, are given in the third section. We also present some empirical data that shows problems that might appear when tactical (project) goals dominate in favor of previously mentioned technical considerations. In fourth section we present some basic concepts of an environment that might help us to integrate different aspects of measurements and to align different levels of goals and measurement.

2. Importance of measurement

Business as well as software project decisions should be based on factual, quantitative information - knowledge that can be obtained only by observing and measuring the products, processes and resources involved. The problem is that there are so many things to measure that is difficult to choose really significant, meaningful and cost effective measurement - for that it is necessary to select those that support business goals of the organization. Although many things can be measured, software development organization rarely use measurements in a systematic way. This can also be concluded from the profiles that summarizes results of different process assessments (ISO, BootStrap, ESI). Usually metrics and measurements scores are low (SPIRE 17,5%, PROCESSUS 5% etc.).

Many companies have established SEPG (Software Engineering Process Group) that coordinates SPI (Software Process Improvement) activities. But before setting-up and starting any activity that is aimed to improve software process and products some fundamental questions should be answered, such as: *Where are we? Where do we want to go? How do we get there? How do we know if we have got to where we wanted to go? How do we compare against competition?*

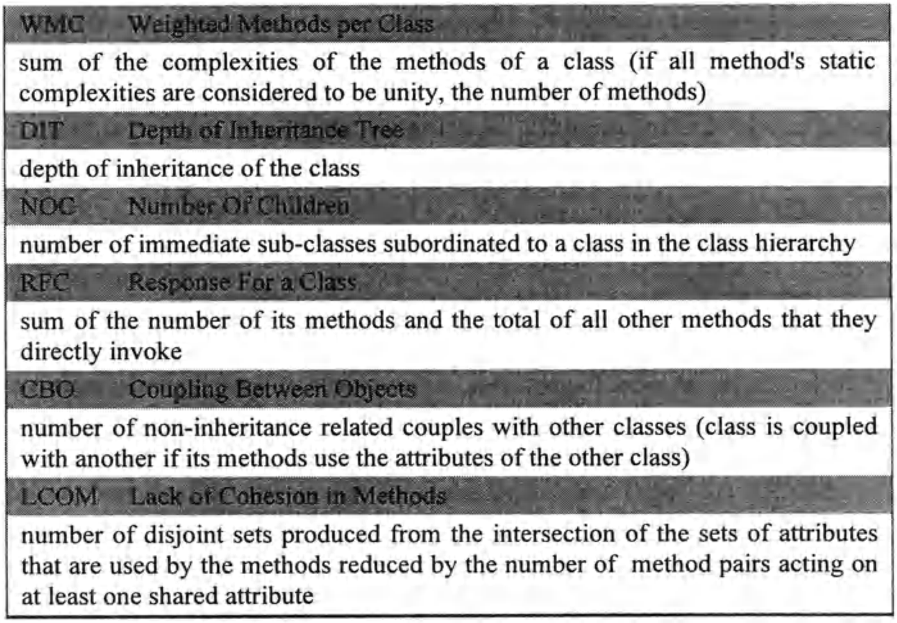
Measures are necessary to identify weakness of the software development process. They also direct corrective activities and enable monitoring the obtained results and effects. In that manner a close loop feedback mechanism is established within which incremental improvements to the software development process can be made over time. Obviously, metrics are a crucial part of SPI activities. Without adequate knowledge and appropriate information we can not assess our stage and define improvement goals. On the other hand, we also need measures and empirical data to investigate if defined goals have been reached.

Introduction of object technology should be considered as SPI activity, since it has been recognized as an enabling factor and it is largely introduced into MIS departments. Introduction of any new technology, methods and/or techniques has always been initiated by desire to increase software quality, provide easier maintenance and achieve higher productivity - regarding object approach this is due to the seamless development, use of same concepts across all phases and higher level of reuse (not only code, but also knowledge, strategies, patterns, test suites). However, the benefits related to object technology can not be achieved and realized without appropriate technical criteria and metrics that direct decisions during software development. Due to some unique principles and concepts that characterize object approach (like inheritance, polymorphism, classes, interfaces) traditional software metrics and evaluation criteria are not very useful and are unsuitable for OO systems, particularly in the existing form. Therefore they should be redefined and there is also a need for completely new set of metrics. The most important metrics suitable for OO systems are presented in the next section.

3. Measures of OO Software

3.1 OO Metrics

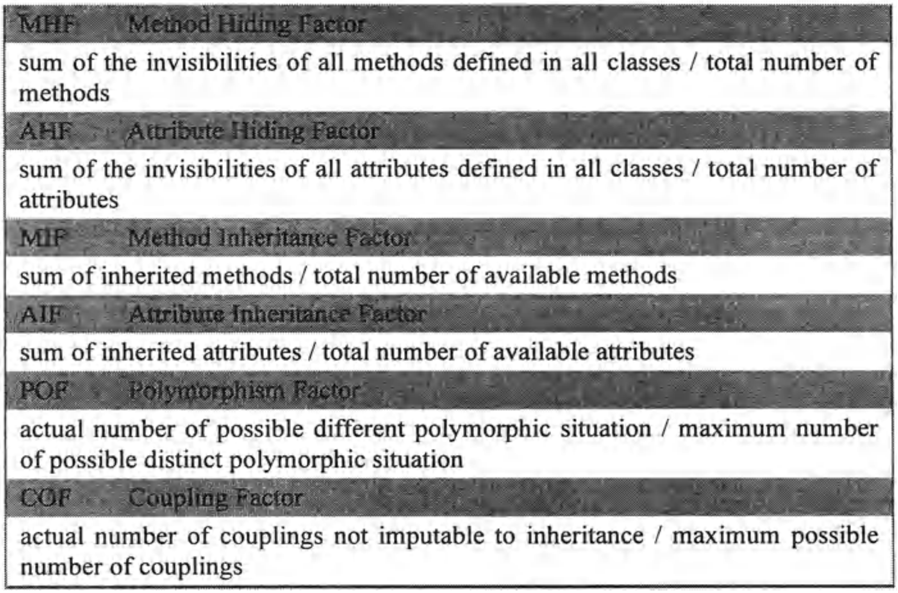
Studies that address metrics suitable for OO development are usually based on and introduce a superset of the metrics proposed by [5] and refined in [6]. They are presented in Table I. However, some deficiencies of these metrics have been identified [7], especially of LCOM (Lack of Cohesion in Methods).



WMC	Weighted Methods per Class	sum of the complexities of the methods of a class (if all method's static complexities are considered to be unity, the number of methods)
DIT	Depth of Inheritance Tree	depth of inheritance of the class
NOC	Number Of Children	number of immediate sub-classes subordinated to a class in the class hierarchy
RFC	Response For a Class	sum of the number of its methods and the total of all other methods that they directly invoke
CBO	Coupling Between Objects	number of non-inheritance related couples with other classes (class is coupled with another if its methods use the attributes of the other class)
LCOM	Lack of Cohesion in Methods	number of disjoint sets produced from the intersection of the sets of attributes that are used by the methods reduced by the number of method pairs acting on at least one shared attribute

Table I: Chidamber and Kemerer's metrics.

MOOD metric suite introduced by [8] could also be useful to evaluate usage of object concepts such as encapsulation (MHF and AHF), inheritance (MIF and AIF), polymorphism (POF) or message passing and association (COF). MOOD metrics are described in Table II.



<table>
<tr><td>MHF</td><td>Method Hiding Factor</td></tr>
<tr><td colspan="2">sum of the invisibilities of all methods defined in all classes / total number of methods</td></tr>
<tr><td>AHF</td><td>Attribute Hiding Factor</td></tr>
<tr><td colspan="2">sum of the invisibilities of all attributes defined in all classes / total number of attributes</td></tr>
<tr><td>MIF</td><td>Method Inheritance Factor</td></tr>
<tr><td colspan="2">sum of inherited methods / total number of available methods</td></tr>
<tr><td>AIF</td><td>Attribute Inheritance Factor</td></tr>
<tr><td colspan="2">sum of inherited attributes / total number of available attributes</td></tr>
<tr><td>POF</td><td>Polymorphism Factor</td></tr>
<tr><td colspan="2">actual number of possible different polymorphic situation / maximum number of possible distinct polymorphic situation</td></tr>
<tr><td>COF</td><td>Coupling Factor</td></tr>
<tr><td colspan="2">actual number of couplings not imputable to inheritance / maximum possible number of couplings</td></tr>
</table>

Table II: Set of MOOD metrics.

Both structured and OO approach rely on encapsulation - the technique by which a set of software components is aggregated into a structure considered as a basic unit from an external point of view. For structured approach, coupling and cohesion are already proven to be useful criteria for evaluating the quality of encapsulation. Since basic units of OO system - objects encapsulate not only a set of methods but also data that represent object's state, coupling and cohesion levels established for structured paradigm are quite useless and they should be redefined. The most systematic and precise taxonomy of coupling and cohesion of OO systems was given by [9].

The volume and growth of research in metrics for OO development indicate metrics have not just become popular but also are important and inevitable to continue efforts in making software engineering an industrial process. Many new metrics have been introduced and some traditional modified (e.g. Function Point Metric to Task Point Metric). Although many metric groups have been established and handbooks developed little empirical data has been published so far (see also [10] and www.sbu.ac.uk/~csse/publications/OOMetrics.html). Some important results and suggestions based on collected empirical data can be found in [11].

Comsoft as a meta-Object Technology Center has developed handbooks on OO metrics for developers and managers [12]. Many useful guidelines how to establish successful reuse program and appropriate reuse metrics are available in [13].

3.2 Empirical data and identified problems

We have developed some metrics data collection tools that enable us to gather metrics data on C++, Smalltalk and Delphi projects (the component for Java is under development). The following commercial projects (products) have been analyzed:

- 10 C++ projects: four business applications (60-80 developed classes), others solve technical problems (30 to 70 classes)
- 3 Delphi project: one for textile industry (50 classes) and two software quality assurance tools (39 and 46 classes)
- Smalltalk projects: modeling tool, object database and complete IS (1454 classes).

Although small, established data collection indicates some assumptions compliant with findings of other researchers, e.g. [11, 12]. However, we do not have enough empirical data to make statistically valid assertions. For that reason only some qualitative interpretation of obtained results will be given and potential problems identified.

Without doubt size metrics such as LOC and no. of classes are not very useful and obviously have no relation with quality attributes. Therefore they should be used very carefully. We should also be careful about no. of methods per class although it might give helpful information when we categorize classes in domains e.g. user interface, business or technical domain and examine distribution of methods by ranges (less than 10, 10-20, 20-30, more than 40).

The average inheritance level of the newly developed classes indicates that Delphi developers and some C++ developers involved in the research do not practice good design strategies that would lead to reusable components. On the contrary, Smalltalk developers and some C++ developers have average depth of inheritance tree from 2,19 to 3,67 (Figure 1). However, it seems that DIT (Depth of Inheritance Tree) is strongly related not only to the developer's experiences in object technology but also to familiarity with particular problem domain. This confirms that reuse does not just happen.

It is quite usual that in small projects OO approach is used for user interfaces design, whereas other parts of the system are built using procedural thinking. For C++ this is also confirmed by a great number of the so called nonmember functions (functions not belonging to any class). Obviously, hybrid OO programming languages, and C++ is one of them, do not enforce OO thinking although they provide mechanisms that support and encourage encapsulation, classes, inheritance and interfaces.

Major violations of encapsulation have been identified for C++ as well as for Delphi developers. Almost half of developers whose projects were analyzed do not respect basic principles of OO approach, e.g. information hiding (Abreau's metrics MHF and AHF). Violation of good design practice, where implementation is hidden from the user of an object, is strongly correlated to the use of OO development methods. One of the findings is that a good OO style of development and a true OO thinking is strongly related with adequate training and developer's comprehension of OO design principles and heuristics.

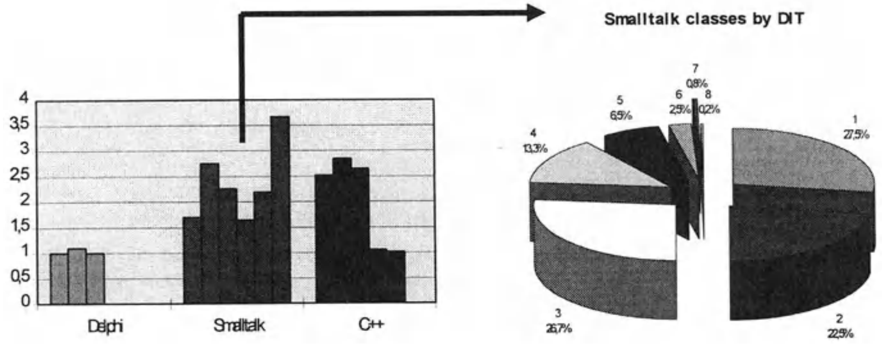


Figure 1. Depth of Inheritance Tree.

Some observations on "software cancer" [14] have also been confirmed. It has been found out that in systems we can quite often find 10% of the classes that embody about 40% of functions and code. And these classes are potentially a major source of problems and majority of the development and maintenance effort is spent on them. Obviously, object technology is not a magic solution by itself. It does not guarantee that the software developed with OO techniques will be better and that the developers use available facilities in the best possible way.

Although many reasons for good object style of development have origins in developers experiences it has to be stressed that many projects (especially those that use Delphi and C++) had been focused primarily on the short-time project goals and were according to Booch classification of projects calendar -driven projects (other types are: requirement driven, quality driven and architecture driven). From the set of analyzed projects, only Smalltalk projects can be characterized as "healthy" - architecture driven projects, since all the time their primary goal was to develop reusable classes organized in component architecture. In addition to the use of patterns and OO development methods appropriate development model (evolutionary) was established. Therefore transition to object technology should be accompanied by SPI efforts aimed to improve process as well as people maturity.

4. Metrics data gathering tools and environments

4.1 Problems with existing tools

As we show in previous section technical measures could be very useful in identifying weakness of our software process and point at rude violations of basic development principles. But for measurement to be efficiently applied during software development an automated support is inevitable. Regarding OO metrics there are some metrics data collecting tools available on the market such as McCabe Visual Quality Toolset (McCabe & Associates), PR:QA C++ (Programming Research), OOMetrics (Hatteras Sofware) but also some tools used only internally such as CPPSTATS and STSTATS (used by IBM for collecting metrics on C++ and Smalltalk). Primarily they are code analysis tools, thus there is still a lack of facilities that might help to collect metrics data also on earlier phases. For that purpose Metrics One (Number Six Soft.) can be used.

tool	supported languages/environments	metrics
OOMetrics (Hatteras Software, Inc.)	focused on IBM's development env. e.g. Visual Age (Smalltalk, C++, Java)	Lorenz&Kidd
iSight++ ™ (IntegriSoft, Inc.)	C++	Chidamber&Kemerer
McCabe Visual Quality Toolset™ (McCabe and Associates)	C++	Chidamber&Kemerer Complexity Metrics
Metrics One (NUMBER SIX Software, Inc.)	only for Rational Rose CASE tool	Chidamber&Kemerer, Lorenz&Kidd
MOODKIT (ISEG/INESC)	C++, Eiffel	MOOD
ObjectMetrics (ObjectSpace, Inc.)	Smalltalk only Visual Smalltalk and VisualWorks	Chidamber&Kemerer some env. specific
Logiscope (VERILOG S.A.)	C++	Chidamber&Kemerer complexity metrics

Table III: OO metrics data collection tools.

Tools (Table III) are usually bound to particular language and/or environment and have single tier architecture what means that metrics are calculated by parser, analyzer. Another problem is that these tools do not enable to collect data on corresponding processes in which measured product has been produced. Regarding development environments and tools it has to be stressed that there is no support and assistance for coupling and cohesion investigation. Despite the fact that determination of all coupling levels can not be automated it is possible to provide some warnings and guidelines, something similar we have at the moment regarding warnings about unused parameters and uninitialized variables. We also believe further integration of modeling and programming tools and introduction of common development repositories will results in wider support and use not only of OO methods but also of OO metrics.

In any case the area of OO metrics evolves much to slow as regard needs of software industry. One of the major problem is that tools do not enable collection of data on corresponding processes in which measured products have been produced. On the other hand there are other sets of tools for inspections, project management, process assessment...

4.2 Integrated environment

Obviously it is necessary to establish an environment that would integrate different measurement tools and help an organization to determine, identify and validate different types of metrics that are suitable and significant for its OO projects. In addition to source code metrics it is necessary to collect all the information on and within software process including static aspects of quality assurance as well as dynamic aspects that are related to particular (individual) project. Calculation of technical metrics should be as much as possible language and phase independent - for that multitiered, repository based environment is required. Environment should act as a meta-gathering environment since it should enable and support addition of new object types, its properties and metrics that should be collected and maintained for that type of objects in the Central Metrics Repository (CMR). Then CMR can enable integration of technical, tactical and strategic goals and offer information that provides better understanding of processes and products.

It is necessary to integrate different kinds of information since people often use data to confirm wrong expectations. Therefore it is inevitable to provide a collection of empirical data and other information related to metrics that might help us to avoid misuse of obtained data and provide relevant information for decision making process. In addition, understanding of and knowledge on applied technology is required. Thus, it is necessary for environment to provide extended context within which adequate measurement can be designed and metrics data utilized to direct software activities toward business goals. Components of such environment should be:

- *Metrics Data Collection Component*

  provides tools and utilities for metrics information collection and extraction: from source code and from development repositories, for product as well as process metrics.
- *Central Metrics Data Repository Management Component*

  aimed to maintain and manage metrics definitions and empirical metrics data.

- *Quality System Documentation Component*

  manages Quality System Documentation (see Figure 2) that also specifies organization long-term business goals as well as policy and procedures how to achieve them.

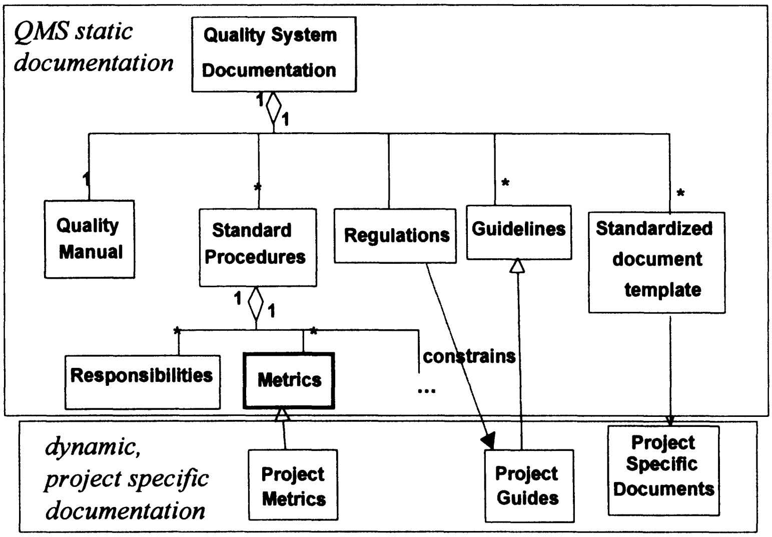


Figure 2. Quality system documentation and some project specific items using UML notation.

- *Process Modeling Component*

  assists in analyzing metrics data, adjusting business goals and providing appropriate development models

- *Project Component*

  assists in setting and meeting achievable commitments regarding cost, schedule, quality and function delivered - compliant with organization business goals including technical aspects and project guides. It should also assist in adjustment when conflict goals emerge.

- *Educational Component*

  enables dissemination of knowledge and information on all aspects of software development; in the case of object technology: tutorials on OOA&D (UML, Objectory, OMT), OO metrics, OO project management, distributed computing, CORBA; COM+, OO testing, object databases etc.)

- *Reuse Management Component*

  provides facilities for managing and searching the reuse repository that should contain not only source code, class libraries, frameworks, test suites, but also other reusable assets, including ideas in the form of patterns for analysis, design etc.

Characteristics and demands of an environment that integrates these components (that clearly interact and utilize services of each other) are therefore:

- repository based integral gathering of metrics information,
- integration of strategic, tactical and technical measurements,
- adjustment of business and technical goals,
- language and phase independent metrics collection,
- collection of source code as well as OO model metrics - "early metrics"
- efficient mechanisms for knowledge and information dissemination
- goal-oriented work (project management, inspections, development)
- reuse-based decisions.

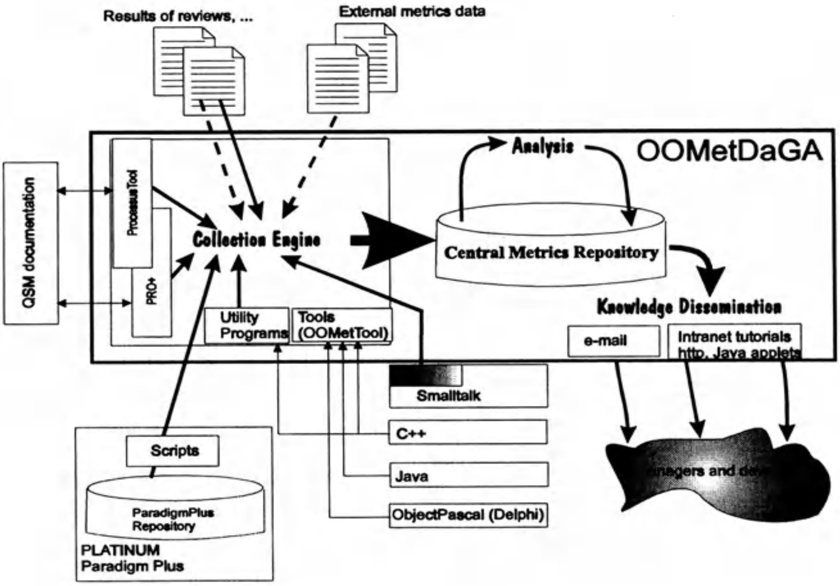


Figure 3. Relations in OOMetDaGa Environment.

4.3 OOMetDaGa environment

Concurrently with the process of defining integrated environment we have developed prototypes of some parts. In the beginning our main goal was to provide an environment that would help to determine, identify and validate metrics that are suitable and significant for OO development. However, afterwards we also decided to collect all information on and within software process, including static aspects of quality assurance as well as dynamic aspects related to particular project. An important feature required is to cross reference concepts, quality system documentation, metric data and project oriented documentation and thus establish integration of static quality system documentation (e.g. quality manual, standard procedures) and dynamic quality issues (specific project documentation). In addition to mechanisms for knowledge dissemination - hypertextual systems with documentation and tutorials and applications for patterns management, OOMetDaGa environment also integrates:

- PROCESSUS*Tool* - supports software process maturity auditing based on the applied methodology (PROCESSUS) for assessment and improvement of software process quality. A detailed explanation of the methodology that is a combination of ISO 9001 and SEI CMM can be found in [2].
- PRO+ - provides support to the software product auditing during acceptance phase according ISO 9126, ISO 9127 in ISO 12119. [3]

Until know we have developed additional tools and components that enable us to collect data from C++, Smalltalk an Delphi projects (Figure 3). A tool for Java is under development. Majority of MOOSE and MOOD metrics are supported. Besides from source code, metrics information can also be obtained from the CASE tool. Scripts have been written (developed) that enable metrics data extraction from Paradigm Plus repository. In that way data on earlier phases and products can also be collected. There are also some utility programs aimed to provide information that is not available in the repository (e.g. cyclomatic complexity and so on). At the moment, this environment is still under development, for the first version of integration we used PowerBuilder and relational database for repository. From the technical point of view we would like to establish a multitired component architecture, to introduce object database and to provide adequate WEB components. We also investigate how intelligent objects might help in realization of the environment. Further extensions with integration with QMS documentation should enable better integration of different goals and measurement and extension with component for Process Modeling will enable to form the necessary support for SEI maturity level 4. Currently, our work is focused on providing support that will enable alignment of the project and business goals both stated in the quantifiable terms. A component under development will support nested GQM (goal-question-metric) technique and a concept of metric independent of any of its multiple implementations [15]. Since goal trees could have subgoals and nested trees that specialize general measures thus providing relation to low level technical measures. Since the same metric could be used in different trees it is necessary to adjust acceptable values and intervals.

5. Conclusion

There is no doubt that it is our common goal to make the software development process an engineering activity. For that we need the ability to assess the quality of development process and resulting products. We need product as well as process metrics since successful control requires some means of measurement.

Characteristics of software products as well as processes should be quantified and analyzed so that performance of activities that produce software items can be predicted, controlled and guided to achieve business and technical goals.

If our goal is to build better software, it is necessary to perform technical OO measurement and relate it not only to tactical but also to strategic goals. We present an environment that might help us to integrate and provide compliance between different aspects of measurement. The environment is helpful in determining, identifying and validating significant metrics for OO development and has the ability to collect information about software process, including static aspects of quality assurance and dynamic aspects related to particular project. It integrates mechanisms for knowledge dissemination, software process maturity and software process auditing. Although software measurement by itself can not solve the problems, it can clarify and focus our understanding of them.

References

1. Whitmire A.S., Object-Oriented Measurement of Software, *Software Encyklopedia*, Vol. 2, John Wiley & Sons, 1994, pp. 737-739.
2. Rozman I., Vajde Horvat R., Györkös J., Heričko M., PROCESSUS - Integration of SEI CMM and ISO Quality Models, *Software Quality Journal*, Vol. 6, No. 1, March 1997, pp. 37-63.
3. Györkös J., Computer Aided Assessment of Software Processes and Products, 1996.
4. Heričko M., Rozman I., Zivkovic A., Knowledge Dissemination as a Basis for Quality Improvement, Knowledge dissemination as a basis for quality improvement. OTS'97 conference proceedings. [s. l.]: Comsoft, 1997, pp. 169-176.
5. Chidamber S. R., Kemerer C. F., Towards a Metrics Suite for Object-Oriented Design, OOPSLA'91 Conf. Proceed., *SIGPLAN Notices*, Vol. 26, November 1991, pp. 197-211.
6. Chidamber S.R., Kemerer C.F., A Metrics Suite for Object-Oriented Design, *IEEE Transactions on Software Engineering*, Vol. 20, No.6, June 1994, pp. 476-493.
7. Hitz M., Montazeri B., Chidamber and Kemerer's Metrics Suite: A Measurement Theory Perspective, *IEEE Transactions on Software Engineering*, Vol. 22, No.4, April 1996, pp. 267-271.
8. Abreu B.F., Carapuca R., Object-Oriented Software Engineering: Measuring and Controlling the Development Process, *Proceedings of the 4th International Conference on Software Quality*, ASQC, McLean, VA, USA, October 1994.
9. Eder J., Kappel G., Schrefl M., Coupling and Cohesion in Object Oriented Systems, University of Klagenfurt, Department of Informatics, Technical Report, 1995.
10. Whitty R., Object-oriented metrics: A status Report, *Object Expert*, Vol. 1, No. 2, Jan/Feb 1996, pp. 35-40.
11. Lorenz M., Kidd J., *Object-Oriented Software Metrics*, Prentice Hall, 1994.
12. ComSoft, *Object Oriented Metrics: a Developer's Handbook, Object Oriented Metrics: a Manager's Handbook*, 1994.
13. Karlsson E.A., *Software Reuse - A Holistic Approach*, John Wiley & Sons, 1995.
14. Haynes P., Detection and Prevention of Software Cancer in OO Systems, Position paper for OOPSLA'96 - Workshop: OO Product Metrics.
15. McGregor J., Managing Metrics in an Iterative Environment, *Object Magazine*, Vol 5, No. 6, October 1995, pp. 65-71.

A Lotus Notes Implementation of A Workflow Automation Tool for ISO 9001 Certification

Yan Ching Ying, Keith C.C. Chan
Department of Computing
The Hong Kong Polytechnic University
Hung Hom, Kowloon, Hong Kong
Email: cscyyan@comp.polyu.edu.hk ,cskcchan@comp.polyu.edu.hk

ABSTRACT

Software quality management involves defining quality goals for software products, establishing plans to achieve these goals, monitoring and adjusting the plans and activities to ensure the software meet the needs of customers. In software development, we believe that it is important to focus on processes and products. As software development involves complex and dynamic interactions of software processes, the lack of a well defined process has thus a direct impact on products' quality. Recently, implementing a quality management system to facilitate the ISO 9001 certification is becoming popular. The essence of the ISO standard is to *'say what you do and do what you say'*. We consider software process as a sequence of tasks that depend on the cooperation between collaborating individuals. In this paper, we describe a system to assist companies to build a QMS and facilitate the ISO 9001 certification. The system has two components : workflow capturing (WC) and workflow enactment (WE). The WC component captures workflow elements and lets users *say what they do.* It adopts three models : i) an actor model, ii) an information model, and iii) a process model. Transformation between process and actor models is supported to provide different views to understand processes and actors. Workflow will then translated into specification. Based on it, the WE component, which is implemented with Lotus Notes 4.6, helps users to *do what they say* and implement workflow by automating task sequences and bringing responsible actors into the process.

1. INTRODUCTION

Software development consists of activities that requires people and resources to be allocated at the right time. This involves complex interactions and dependencies among activities and people. To avoid problems such as project delay and failure to meet requirements, much emphasis has been put on international quality standards, such as the ISO 9001, and software quality management (SQM). To comply with the ISO 9001, organizations should conform to requirements by *saying what they do* and *doing what they said* [1]. As software is subjected to frequent changes, it is difficult to document what and how processes and staff do.

C. Hawkins et al. (eds.), *Software Quality Management VI*

Therefore, to fulfill ISO requirements and build a SQM, a comprehensive process model is required so that people can easily describe and understand what to do.

Our system helps in capturing and enacting processes, and provides quality templates for documentation. The WC component provides graphical interfaces for users to describe workflow. It supports i) an actor model that captures organizational structure, ii) an information model that records presentation details, iii) a process model which supports workflow automation. Also, transformation between process and actor models is allowed to provide multi-perspectives when capturing and understanding processes and the interaction between actors. This can fulfill *'say what to do'* in the ISO philosophy. Workflow details is translated into specification with reference to the WIDE workflow specification. Based on these, the WE component ensures consistent process implementation across the organization by implementing procedures through groupware. It takes advantage of useful groupware features to streamline communication and interactions in processes. Also, it provides status tracking of processes and tasks. This helps companies to *'do what they say'*. Our system has been tested successfully by stimulated cases. And currently, it is used to implement a process used in a large bank to handle its new service request.

2. RELATED WORKS

In the past decade, there are many software products for QMS and ISO 9000 certification. We have studied twenty-two of them. Their features are grouped into five categories : i) those that assists QMS documentation, such as the provision of templates for quality manuals and procedures; ii) those for document control, such as the provision of forms and report generation; iii) those to assess ISO readiness, such as the provision of checklists; iv) those that facilitate system audit, such as the maintenance of audit schedule; and v) others such as network-enablement and on-line help. Although these features are useful for ISO certification, none of the tools surveyed contain all of them. Also, only three of them are developed for the software industry but unfortunately not all clauses are included. Therefore, there is currently a lack of comprehensive ISO tools in the market for the software industry.

The major products in the workflow market are Lotus Notes, IBM FlowMark, InConcert by Xerox and ActionWorkflow by Action Technologies. Lotus Notes is document and database oriented and workflow involves document routing and electronic mails. FlowMark is an object-oriented client-server system which is not a stand-alone product and access to database is not modeled. A backend database is required. InConcert is data-oriented and it does not address actor relationship and quality management. Lastly, ActionWorkflow is based on speed act that focus on communication patterns between actors. It is not developed for task completion and is too complex to model software processes. Therefore, Lotus Notes is better because it has rich build-in database capabilities and electronic messaging functions.

3. WORKFLOW CAPTURING

Our workflow model is derived from the WIDE model from the Workflow Management Coalition (WfMC). With some enhancements, our system captures workflow by the information model, the actor model and the process model. Transformation between process and actor model is allowed. Our model is unique as it has more focus on task relationships and the relationships between people.

3.1 Information Model

This model identifies information objects (useful data) involved in workflow. The objects are defined as forms which are characterized by : form name, description and procedures involved. Owing to the interactive nature of workflow, it involves document routing and much information is presented by forms. Workflow definition and control data are stored to facilitate process flow, such as the process definition, organization data, and work-to-do list. In our system, forms are the templates to create documents in the QMS and to record the actual level of quality performance. After defining forms, they can be linked to tasks, where the information is used by the workflow participants (actors).

3.2 Actor Model

For each actor, we specify his department, subordinates, superordinates and role. Our system allows three types of actor relationship to be defined : information sharing, task cooperation and task dependency. Information sharing is document referencing among actors. It is the time for actors to retrieve information to accomplish certain tasks. For example, a system specification is shared among system analysts and programmers. So, referencing the same document implies actors are doing the same or related tasks. When several actors work in the same task, for example, both project manager and system analyst participate in project planning and scheduling, we consider they have task cooperation relationship. If an actor cooperates with many actors, it may imply an uneven workload or insufficient human resources. Since a task can start only after the completion of its preceding task(s), when actor(s) in the coming task(s) can start working depends on when the actor(s) in the preceding task(s) finished their works. We consider the actor(s) in the coming task depend on actor(s) in the preceding task(s). For example, the programmer can start coding if the system analyst has finalized the design. Task dependency has different scope from task cooperation. Task cooperation focuses on actors in a task while task dependency addresses the sequence among consecutive tasks.

3.3 Process Model

Process is modeled as a combination of tasks with defined sequences. Task is the building block in this model. Apart from basic attributes such as task name and description, runtime task attributes include type (manual, automatic), maximum time allowed, time unit (day, hour, minute), form created, modified and referenced, actors responsible for the task as well as actors that will be notified and method to handle exceptions (ignore, stop immediately, send error message, start recovery task). Runtime evaluation of attributes allows tasks to be dispatched dynamically. Manual tasks must have actor(s) to perform the task and automatic tasks rely on an agent, an automatic design element, to perform the task. Our system navigates a process by evaluating task relationships. It allows three types of task relationship to be defined : i) sequential, ii) parallel and iii) conditional.

Sequential relationship involves consecutive task execution. A series of sequential relationship forms a path in a process. It represents the order of tasks and indicate the flow of work. For tasks A and B connected sequentially, task B can start only if task A completes. And, parallel relationship involves simultaneous task execution that several tasks will be in active. Executing one task does not depend on the others and will not affect their executions. If task A is connected to task B and C in parallel and task D follows, both task B and C will start after task A accomplishes. And, task D can be initiated after both task B and C have been completed. On the other hand, conditional relationship selects execution path based on conditions. Although there are multiple execution paths, only one of them can be in active based on the evaluation of conditions.

3.4 Transformation of Process and Actor Models

In a process model, adding actors in tasks creates task cooperation relationship and defining form referenced builds information sharing relationship. If there are several actors in a task, they have task cooperation relationship. When several actors reference the same form, they have information sharing relationship. In task cooperation, connector links actors having task name as attribute. And, in information sharing, connector attributes are the referenced form name and the task names that the actors reference the form. These relationships are non-directional but task dependency is directional. That is, actors in the succeeding and preceding tasks must be specified. Task dependency is unconditional in sequential and parallel task relationships. In conditional task relationship, based on the conditions, different tasks will be followed. Then, task sequence will be changed, and connector attributes are thus varied. Condition can be specified by agent (Notes design element) and form value. Using agent, agent name and name of next task are needed. After running the agent, it returns a true/false value to determine whether the next task can start. For example, if the agent 'CheckResource' returns a true value, task 'NotifyActors' will start. Alternatively, using form value, form name, field name, operator (includes, =, <>, >, <, >=, <=), field value and name of next task are required. These parameters are used to check if a field in a form satisfies a value so that the next task can be followed. For example : if ***field*** *'severity'* in ***form*** *'problem report'* ***equals*** *(=)* to *high*, then ***task*** *'report to project manager'* starts. Different conditions can be specified to control the execution path. In the unconditional case, connector specifies which actor depends on which actor and the names of preceding and succeeding tasks. In the conditional case, additional attributes such as condition for that dependency is required. An actor model can be built from a process model as summarized in Table 1 below.

Task creation	Task connection
Assigning actors ⇒ Task cooperation **Actor 1** Task1, Task2 **Actor 2**	Unconditional task relationships ⇒ Task dependency **Actor 1** **Actor 2** Task 1 → Task 3
Defining form referenced ⇒ Information sharing **Actor 1** form 1 **Actor 2** Task 1 form 2 Task 2, Task 4	Conditional task relationships ⇒ Task dependency **Actor 1** conditon **Actor 3** Task 1 statement → Task 4

Table 1 Transformation logic from task to actor model.

In the actor view, defining task cooperation and information sharing relationships captures task properties. Actors are determined from task cooperation and form referenced is defined by information sharing relationship. When actors have task cooperation relationship, they are responsible for some common task. If they have information sharing relationship, they reference the same form in the same or different task. Our system uses task dependency relationships to determine task sequences and conditional branches. Unconditional task dependencies ensure restricted task sequence. And, conditional task dependency provides possible task dependency. In different instances of the same process, different condition may be satisfied which leads to different task. Task sequences are then varied. Fig. 2 is the process model deduced from Fig 1. Actor 2 depend on Actor 1 as there is a sequential task relationship between Task 1 and 2. Based on different conditions, either Task 3 or 4 may follow Task 2. If condition 1 is true, Actor 3 will depend on Actor 2. If condition 2 is true, Actor 4 will depend on Actor 2. Connector attributes in actor model are similar to the process model. Having all the task properties, sequence and possible branching, our system can transform an actor model to a process model. Table 2 summarizes this transformation algorithm.

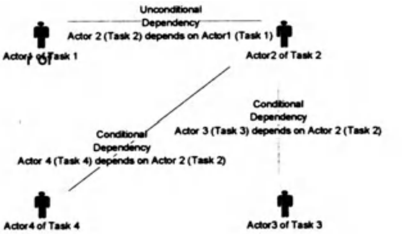

Unconditional
Dependency
Actor 2 (Task 2) depends on Actor1 (Task 1)
Actor1 of Task 1
Actor2 of Task 2
Conditional
Dependency
Actor 3 (Task 3) depends on Actor 2 (Task 2)
Conditional
Dependency
Actor 4 (Task 4) depends on Actor 2 (Task 2)
Actor4 of Task 4
Actor3 of Task 3

Fig. 1 An illustrative example of actor model.

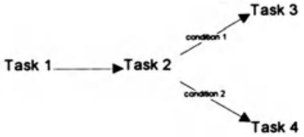


Fig. 2 An illustrative process model.

Information sharing	Task cooperation	Task dependency
Actor 1 —Form 1 / Form 2— Actor 2 Task 1 Task 1, Task 4 Actor 1 is an actor in Task 1 and Actor 2 is an actor in Task 2 and 4. Form 1 and 2 are the referenced forms of Task 1, 2 and 4.	Actor 1 Task 1, Task 2 Actor 2 Actor 1 and 2 are one of the actors in Task 1 and Task 2.	Actor 1 ——▶ Actor 2 Task 1 Task 3 Actor 1 is an actor in Task 1 and Actor 2 is an actor in Task 3. Task 1 can starts only after Task 3. Actor 1 —condition statement▶ Actor 2 Task 1 Task 3 Actor 1 is an actors in Task 1 and Actor 2 is an actors in Task 3. Task 1 starts after Task 3 if the condition is satisfied.

Table 2 Transformation logic from actor to process model.

4. WORKFLOW SPECIFICATION

4.1 The Actor Specification

Our specification uses ACTOR definition to capture : i) *name,* a Notes user name; ii) *role,* position of the actor; iii) *department,* department which the actor works for; iv) *subordinates,* actor names of the subordinates; and v) *superordinates,* actor names of the superordinates. Attributes are assigned to a data type and actors are assigned to a role, captured by the ROLE definition. Each role has a name and description. The actor model can be populated using the REGISTER ACTOR statements. Table 3 is an example of the actor specification.

```
ACTOR_MODEL DEVELOPMENT
        ACTOR HAS
                name : STRING;
                role : STRING;
                department : STRING;
                subordinates : LIST OF STRING;
                superordinates : LIST OF STRING;
        END_ACTOR;

        ROLE Project Director
                desc : Manage and Control Projects;
        END_ROLE;

        ROLE Test Manager
                desc : Unit and System Test;
        END_ROLE;
END_MODEL

REGISTER ACTOR OF DEVELOPMENT
        (name, role, department, subordinates, superordinates)
        [(Ying/POLYU, Test Manager, ITS, [] , [Keith Chan/POLYU] ) ,
        (Keith Chan/POLYU, Project Director, Computing,[Ying/POLYU] , [] ) ];
```

Table 3 Actor Specification.

4.2 The Information Specification

The FORM definition is used to capture : i) *name*, form name; ii) *description*, description of it; and iii) *procedures*, procedure names that will use this form. In Notes, form design is hidden from other applications. It is so far impossible to obtain the layout of a Notes form and export to the other applications. So, our information specification is different from the WIDE information specification. Our information model can also be populated using the REGISTER statements. Table 4 shows the form specification used in the design process.

```
INFORMATION_MODEL Design
          FORM HAS
                    name : STRING;
                    description : STRING;
                    procedures : LIST OF STRING;
          END_FORM;
END_MODEL

REGISTER FORM OF Design
          (name, description, procedures)
          [(Functional Specification,Detail description of functions,[Design Control,Inspection and Testing]) ,
           (Technical Specification,Detail specification,[Design Control,Statistical Techniques] )];
```

Table 4 Information Specification.

4.3 The Process Specification

For each process, our specification capture : name, description and name of database used in the process. Respective actor and information models have to be imported into this specification using the USES statement. The START statement is used to specify the first task in the process. Task is specified by TASK definition, our specification captures GENERAL information such as : name, description, type, maximum execution time allowed, and time unit used. Other information will also be captured, such as : i) *ACTOR,*actor names for the task; ii) *NOTIFY,* actor names to be informed; iii) *FORM_CREATE,* form names that will be created; iv) *FORM_MODIFY,* form names that will be modified; v) *FORM_REF,* form names that will be referenced; vi) *EXCEPTION,* method to handle exception and respective argument; and vii) *COMPLETE,* how to proceed upon task completion. Since they are four ways to proceed upon task completion (End Process, Start Single Task, Start Single Task Based on Condition and Start Multiple Tasks), the number of arguments of COMPLETE is varied. No argument is required for End Process (Entry 1 in Table 5). For Start Single Task and Start Multiple Task, task name(s) is required (Entry 2 and 3 in Table 5). The number of arguments in Start Multiple Task depends on the number of tasks that run in parallel. For Start Single Task Based on Condition, the number of conditions and the condition statements are required. Conditions are specified by agent or form value. They are included in a pair of bracket (Entry 4 in Table 5). An example of workflow specification is shown in Table 6.

	Statements
1.	*COMPLETE (method, argument) [End Process,]*
2.	*COMPLETE (method, argument) [Start Single Task, Approve change request]*
3.	*COMPLETE (method, argument) [Start Multiple Task, Schedule change request, Schedule UAT]*
4.	*COMPLETE (method, argument) [Start Single Task Based on Condition, 2,* *(By Form, Change Request Form, status, =, disapproved, Archive change request),* *(By Form, Change Request Form, status, =, approved, Impact analysis)]*

Table 5 Complete Statements under different conditions.

```
WORKFLOW_MODEL Software Change Request (name, description, database)
[Software Change Request, Standard procedures to handle change request, wf\demo]
USES ACTOR_MODEL ReqActor;
USES INFORMATION_MODEL ReqForm;
START Initiate change request;

TASK Initiate change request
   GENERAL (description, type, precede, time, unit) [test, , 1, 30, Minute]
   ACTOR (actor list) [User]
   NOTIFY (actor list) []
   FORM_CREATE (form list) [Change Request]
   FORM_MODIFY (form list) []
   FORM_REF (form list) []
   EXCEPTION (method, argument) [Ignore, ]
   COMPLETE (method, argument) [Start Single Task, Approve change request]
END TASK;

TASK Approve change request
   GENERAL (description, type, precede, time, unit) [test, , 1, 2, Day]
   ACTOR (actor list) [Project Manager]
   NOTIFY (actor list) []
   FORM_CREATE (form list) []
   FORM_MODIFY (form list) []
   FORM_REF (form list) [Change Request]
   EXCEPTION (method, argument) [Ignore, ]
   COMPLETE (method, argument) [Start Single Task Based on Condition, 2, (By Form, Change Request,
   status, =, not ok, Archive change request), (By Form, Change Request, status, =, ok, Impact analysis)]
END TASK;

TASK Archive change request
   GENERAL (description, type, precede, time, unit) [test, , 1, 2, Day]
   ACTOR (actor list) [Support]
   NOTIFY (actor list) []
   FORM_CREATE (form list) []
   FORM_MODIFY (form list) []
   FORM_REF (form list) []
   EXCEPTION (method, argument) [Ignore, ]
   COMPLETE (method, argument) [End Process, ]
END TASK;

TASK Impact analysis
   GENERAL (description, type, precede, time, unit) [test, , 1, 4, Day]
   ACTOR (actor list) [Support]
   NOTIFY (actor list) []
   FORM_CREATE (form list) []
   FORM_MODIFY (form list) []
   FORM_REF (form list) [Change Request]
   EXCEPTION (method, argument) [Ignore, ]
   COMPLETE (method, argument) [Start Multiple Task, Schedule change request, Schedule UAT]
END TASK;
```

Table 6 Process Specification.

```
TASK Schedule change request
   GENERAL (description, type, precede, time, unit) [test, , 1, 1, Day]
   ACTOR (actor list) [Project Manager]
   NOTIFY (actor list) []
   FORM_CREATE (form list) []
   FORM_MODIFY (form list) []
   FORM_REF (form list) [Staff schedule, Project schedule]
   EXCEPTION (method, argument) [Ignore, ]
   COMPLETE (method, argument) [Start Single Task, Do the changes]
END TASK;

TASK Schedule UAT
   GENERAL (description, type, precede, time, unit) [test, , 1, 1, Day]
   ACTOR (actor list) [Project Manager]
   NOTIFY (actor list) []
   FORM_CREATE (form list) []
   FORM_MODIFY (form list) []
   FORM_REF (form list) [Staff schedule]
   EXCEPTION (method, argument) [Ignore, ]
   COMPLETE (method, argument) [Start Single Task, Do the changes]
END TASK;

TASK Do the changes
   GENERAL (description, type, precede, time, unit) [test, , 2, 10, Day]
   ACTOR (actor list) [Programmer, Support]
   NOTIFY (actor list) []
   FORM_CREATE (form list) []
   FORM_MODIFY (form list) []
   FORM_REF (form list) [Change Request, Source code]
   EXCEPTION (method, argument) [Ignore, ]
   COMPLETE (method, argument) [Start Single Task, Test the changes]
END TASK;

TASK Test the changes
   GENERAL (description, type, precede, time, unit) [test, , 1, 8, Day]
   ACTOR (actor list) [Programmer, Tester]
   NOTIFY (actor list) []
   FORM_CREATE (form list) []
   FORM_MODIFY (form list) []
   FORM_REF (form list) [Source code, User requirements]
   EXCEPTION (method, argument) [Ignore, ]
   COMPLETE (method, argument) [Start Single Task, Perform UAT]
END TASK;

TASK Perform UAT
   GENERAL (description, type, precede, time, unit) [test, , 1, 5, Day]
   ACTOR (actor list) [User]
   NOTIFY (actor list) []
   FORM_CREATE (form list) []
   FORM_MODIFY (form list) []
   FORM_REF (form list) [Change Request]
   EXCEPTION (method, argument) [Ignore, ]
   COMPLETE (method, argument) [End Process, ]
END TASK;
```

Table 6 (Cont'd) Process Specification.

5. SYSTEM IMPLEMENTATION

5.1 The Workflow Capturing (WC) Component

This component is implemented by Visual C++. A graphical workflow editor is used to define workflow and its elements. For a new process, a name, description and database name, which contains the workflow logic and form designs, are required. Based on the actor and form specifications generated in the WE component, actors and forms are imported to define workflow. Fig. 3 shows a previously defined process. The left hand panel shows workflow elements and the right hand panel is the task view of the process.

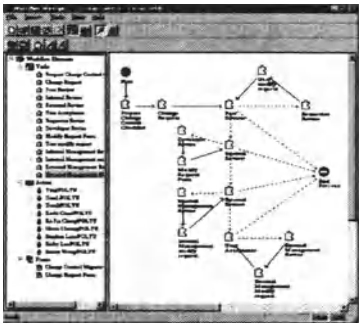


Fig. 3 A defined change request process.

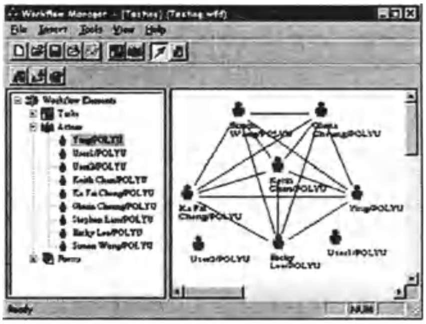
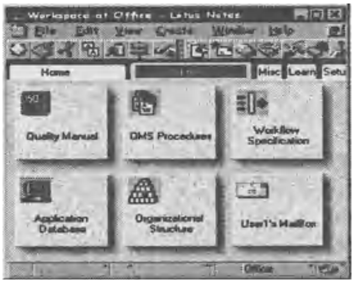


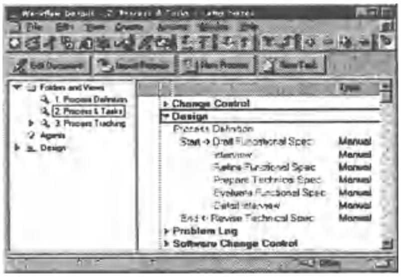
Fig. 4 An example of actor view.

A new task has attributes : (i) "General" (task name, description, task type and maximum duration); (ii) "Forms Created" (forms to create); (iii) "Forms Modified" (forms to modify); iv) "Forms Referenced" (forms to reference); v) "Actors" (responsible actors); and vi) "Exception Handling" (how task proceeds when exception is raised). Once entered, a new task icon will be added in the right panel. Unconditional and conditional task relationship are used to specify flow of work. No property is required in unconditional relationship. In conditional relationship, the conditions which the next task should be executed is needed. For example, a condition can be evaluated by a form 'DCR' and if the field 'continue' equals (=) to 'adopt', the specified next task will start. Users can swap between the task and actor views. The actor view has icons corresponding to the actors on the left hand panel (Fig. 4). Actor properties includes : i) "General" (actor name, title, department and task involved); ii) "Subordinates" (actors that is the subordinate of the current actor); and iii) "Superordinates" (actors that is the superordinate of the current actor). Actor icons are connected if they are defined in the task view to i) be responsible for the same task; ii) depend on the other; or iii) reference the same form(s). Users can check if actors have any relationship between them and if so, how they are related. Actor relationships will be changed accordingly when the properties or sequence of tasks changes in the task view. After the workflow is defined, the respective process specification can be generated and used in the workflow enactment component.

5.2 The Workflow Enactment (WE) Component

The WE is implemented on top of Lotus Notes 4.6. It consists of six databases (Fig. 5) : quality manual, quality procedures, organizational structure, workflow specification, application database and user's mailbox. The quality manual database includes quality manual and the QMS procedures database has procedures in the organization. The organizational structure database includes actors in the organization and actor specifications can be generated from it. Workflow defined in the WC component is imported in the workflow specification database (Fig. 6). Only process administrator (a special role defined in the database) can do this.



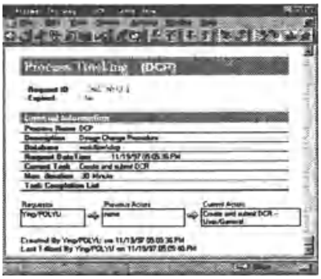


- Folders and Views
  - 1. Process Definition
  - 2. Process & Tasks
  - 3. Process Tracking
  - Agents
- Design

▸ Change Control	
▾ Design	
Process Definition	
Start → Draft Functional Spec	Manual
Interview	Manual
Refine Functional Spec	Manual
Prepare Technical Spec	Manual
Evaluate Functional Spec	Manual
Detail Interview	Manual
End ← Revise Technical Spec	Manual
▸ Problem Log	
▸ Software Change Control	

Fig. 5 Databases in the WE Component in Notes. **Fig. 6** Workflow Specification Database.

Process can be initiated by process owner (another role of the database). A process tracking document (a Notes document) that contains all the necessary information to start the process will be generated (Fig. 7). Each responsible staff will receive a task notification message which includes links for users to go directly to the process, task definitions, and the process tracking document. Through the process tracking document, actors can mark task to be completed, view previous comments, and raise exception. Also, actors can create, modify and reference forms. However, only current actors that have not finish their tasks are authorized to perform these functions. The system waits until all the responsible actors have completed their works. In order to start next task(s), all the forms required to create in the current task must be created. The WE component keeps track of the status of all processes that has been initiated in the organization, sorted and categorized by the current responsible actors, process name or request date (Fig. 8).



Process Tracking (DCP)

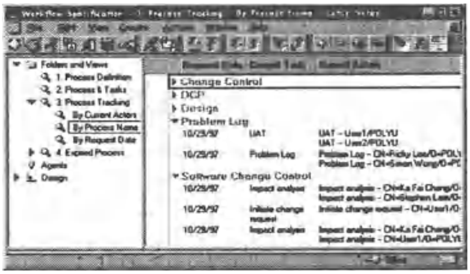
Request ID
Expired

General Information

Process Name	DCP
Description	Design Change Procedure
Database	workflow\dcp
Request Date/Time	11/15/97 05:05:36 PM
Current Task	Create and submit DCR
Max. duration	30 Minute
Task Completion List	

Requestor: Ying/POLYU → Previous Actors: none → Current Actors: Create and submit DCR – User/General

Created By Ying/POLYU on 11/13/97 05:05:36 PM
Last Edited By Ying/POLYU on 11/15/97 05:05:40 PM


Workflow Specification - 3. Process Tracking \ By Process Name - Lotus Notes
File
Edit
View
Create
Actions
Window
Help
Request Date
Current Task
Current Actors

- Folders and Views
  - 1. Process Definition
  - 2. Process & Tasks
  - 3. Process Tracking
    - By Current Actors
    - By Process Name
    - By Request Date
  - 4. Expired Process
  - Agents
- Design

▸ Change Control		
▸ DCP		
▸ Design		
▾ Problem Log		
10/29/97	UAT	UAT – User1/POLYU UAT – User2/POLYU
10/29/97	Problem Log	Problem Log – CN=Ricky Lee/O=POLY Problem Log – CN=Simon Wong/O=PC
▾ Software Change Control		
10/29/97	Impact analysis	Impact analysis – CN=Ka Fai Chang/O- Impact analysis – CN=Stephen Lee/O-
10/29/97	Initiate change request	Initiate change request – CN=User1/O-
10/29/97	Impact analysis	Impact analysis – CN=Ka Fai Chang/O- Impact analysis – CN=User1/O=POLYU

Fig. 7 Sample process tracking document. **Fig. 8** The process tracking view.

6. CONCLUSION

The major purpose of installing a SQM system in an organization is to improve the quality of the software processes. And, the basis for a SQM system will best be based on some international quality standards as this will give the confidence of its successful implementation and limit risk. In this paper, a workflow approach is used to manage processes. We have developed a system that is able to make process visible through well defined workflow elements, structure and representation to support group work. It helps actors to cooperate and communicate, and assists companies to install SQM system and facilitates the certification process of the ISO 9001.

The system has components which are responsible for different functions. With modification to the WIDE workflow model, our system provides a rich conceptual model to capture processes and support features that are required. In the implementation, we employ object oriented technology to model tasks, actors and processes for workflow capturing. In workflow enactment, groupware technology is used to support communication, enhance collaboration and facilitate cooperation of software development team members. In conclusion, in order to build a SQM system and improve software quality, we particularly focus on improving processes by automating and managing workflows with the help of groupware.

7. References

1. Gianluigi Caldiera, Impact of ISO 9000 on Software Maintenance, In : Proceedings of Conference on Software Maintenance, CSM-93, 1993, pp. 228 - 230.

2. Berthold Reinwald, C. Mohan, Structured Workflow Management with Lotus Notes Release 4, In Digest of Papers. *COMPCON '96.* Technologies for the Information Superhighway. Forty-First IEEE Computer Society International Conference, 1996, pp. 451 - 457.

3. Christoph Bußler, Stefan Jablonski, An Approach to Integrate Workflow Modeling and Organization Modeling in Enterprise, In Proceedings. Third Workshop on Enabling Technologies : Infrastructure for Collaborative Enterprises, 1994, pp. 81 - 95.

4. The Workflow Reference Model. In Technical Report TC00-1003, Workflow Management Coalition, Nov 1994.

5. F. Casati, P. Grefen, B. Pernici, G. Pozzi, G. Sanchez, WIDE Workflow model and architecture, Workflow Management Coalition, April 96.

6. Jim Welsh, Jun Han, Software Documents : Concepts and Tools, Software : Concepts and Tools, Vol. 15, 1994, pp. 12-25.

Section 5

Quality Issues

Automatic Software Evaluation Based on Domain Knowledge and Formal Evaluation Specification

Boris I. Cogan, Tamara O. Matveeva
Institute for Automation & Control Processes
Far Eastern Branch of the Russian Academy of Sciences
5 Radio Str., Vladivostok, 690041, Russia
cogan@iapu2.marine.su

Abstract

The paper presents the language–oriented approach to software evaluation. The approach implies: (1) creation of a software system having general knowledge of the domain 'software evaluation'; (2) development of a formal evaluation specification for any specific software in terms of the knowledge represented in a special–purpose problem–oriented language; (3) generation of an ad hoc knowledge–based evaluation workbench basing on the system and the specification.

1 Introduction

Nowadays practice of software evaluation has problems that are due to lack of well-defined definitions of some notions used to describe software evaluation process. Many definitions of software quality (SQ) models and metrics, software product (SP) models and methods to build them, and assessment methods are informal [1]. So, it is not always possible to obtain repeatable and reproducible evaluation results when different tools are used [2].

On the other hand, any workbench used in evaluation practice has strict definitions of all notions, but, usually, the definitions are not available for evaluators.

The paper presents the language–oriented approach (LOA) to SQ evaluation, which is intended for creation of formal, readable and 'executable' evaluation specifications. The step–by–step execution of such a specification produces the software evaluation result. Main principles to create the specification are considered.

The approach has been developed at the Institute for Automation and Control Processes for about ten years. In English, it is presented in the papers [3-8]. Now this work is supported in part by The Royal Society (The joint British–Russian project "Language–oriented approaches to software assessment", 1996-1998).

C. Hawkins et al. (eds.), *Software Quality Management VI*

2 The Starting Point

The idea to use knowledge–based techniques in SQ evaluation is not new. The most known work based on the idea is the project SCOPE – Software CertificatiOn Programme in Europe, 1989-1993. One of its most important result is 'The formal model of software evaluation and certification' as well as the automated evaluation technology based on the model [2, 9]. H.-L.Hausen and D.Welsel made the main contribution to the model and refined it later.

However, 'the formal model' and the technology have some shortcomings.

1. Insufficient understandability of the model. The main components of the model are: (a) SP characteristics and metrics, (b) SP and software process information, (c) evaluation methods and tools, and (d) their interrelations. All the components are specified by using a first order predicate calculus language (Prolog was used in the implemented version of the model). Such a specification is not high understandable and maintainable. Practice of development of knowledge–based systems (KBSs) has showed that the application of rule–based representation has some limitations when a lot of objects and many types of their relations are defined [10].

2. Vague semantics of some notions used. A specific evaluation process is defined in terms of metrics, measurement and assessment methods, etc. A program module, 'brick' or 'evaluation module', corresponds to an evaluation technique including metrics, pass–fail criteris, etc. Semantics of some bricks is rather vague because basic notions of attributes, models, etc., encapsulated in the bricks are treated in different ways in the literature. A user of the technology cannot know, which of the treatment has been used to implement a brick. So, he/she uses vague notions to define his/her evaluation process. To help the user, the intelligent advisor was developed. However, a trial of the advisor showed that its advice is too general and subjective [2].

The approach of this paper permits avoiding the shortcomings.

3 Language–Oriented Approach

The main aim of LOA is production of rigorous, semantically valid definitions of notions of the domain, a proper combination of the notions in the system, and application of the system to evaluate specific software. The system would support relevant international standards and is in agreement with the well–known and often–quoted papers and books.

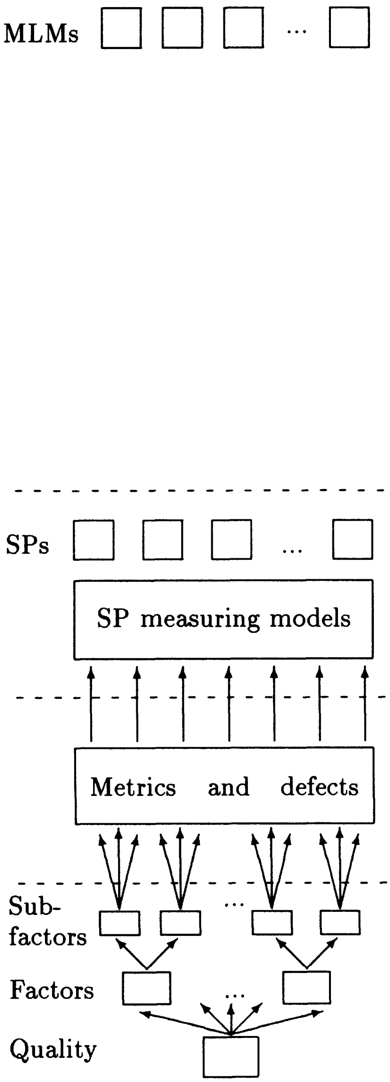
LOA consists in use of three levels of concordant formal languages, which are intended for definition of (Fig.1):

(1) the basic attribute set of symbols and constructions of any SP writing language. The set is built according to semantics and syntax of this strictly defined language (in particular, a programming language).

Here, the *measuring language model* (MLM) and the *measuring language processor* (MLP) are developed for any language used. MLP is a version of

Relationships among SQ model, MLMs, SP representation languages, and MMs of the SPs

Activities for SQ evaluation when the language-oriented approach is used



Level 0. Occurring once activities done by developers of software development and assessment tools.

1. Developing SP writing languages and their MMs.
2. Developing MLP for any language used.
   A generalized representation of MMs of SP written in different languages.
3. Developing a language to define SP metrics and defects.
4. Developing a processor of the language.
5. Formal defining the metrics and defects.
6. Developing a language to declare SQ requirement specifications and software measurement and assessment (i.e., a metrics framework declaration language).
7. Developing a processor of the language.

Level 1.

1. Developing specific SPs.
2. Measuring all SPs in terms of MLMs used;
   The single measuring tool, MLP, for each SP writing language.

Level 2.

1. Measuring all SPs in terms of SP metrics and defects basing on SP MM.
   The single measuring tool for all SP metrics and defects.

Level 3.

1. Formal declaring quality requirement specification for a specific software.
2. Executing assessment.
   The single knowledge-based task-oriented assessment tool.

Figure 1. Relationships among the set of formal languages and their processors.

the commonly used language processor, which additionally generates the SP measurement result, so called SP *measuring models* (MMs), in accordance with the MLM [5, 6, 8]. MM of a specific SP includes values of all static (and dynamic) attributes of all SP symbols and constructions (in context of the number of executions of the SP being a program);

(2) SP metrics in terms of the MLMs. Under this definition, any metric has rigorous definition in terms of semantics of the languages;

(3) knowledge of the domain 'software quality' and the quality requirements specification for specific SPs in terms of formally defined SP metrics and defects as well as measurement theory.

Here, all 'high-level', external (in relation to the SPs) notions of the domain are uniquely defined in terms of 'low-level', internal notions.

Then the metrics framework used in a specific evaluation process may be defined formally in term of (uniquely defined) SP metrics according to the universally recognized measurement theory. SP quality requirements specification may be expressed in terms of notions defined in the metrics framework. After these, a knowledge representation language processor may draw a decision on the acceptance or rejection of any evaluated SP using the following information:

- the quality requirements specification, as an executed program,
- a set of metrics values obtained under SP measurement, as its data.

Thus, application of the LOA is a sequential, 'level-by-level' usage of the set of formal languages that provide obtaining unambiguous intermediate and final results of all activities done (Fig.1). The levels are: (1) specifying quality requirements to a specific software, (2) measuring SPs of the software, and (3) evaluating this software or its part.

The examples of use of the languages are presented in the rest of the paper.

4 Principles of Knowledge Representation

Here, the following principles of knowledge representation are used.

1. Declarative knowledge is separated from procedural one. The declarative knowledge is definitions of internal and external SP attributes (SQ factors and subfactors, SP metrics) and their interrelations, i.e., sets of notions. There is a set of attributes connected with any notion (their interrelations are considered as attributes as well). So any notion is written as a relation, i.e., as a named ordered list of attributes $O(A_1, A_2, ..., A_n)$. A scale is defined for any attribute as: (1) the type of the scale, (2) the values range, and (3) strict rules to get the values. The rules may be defined by one of two ways.

(a) By functions (formulas) over attributes of different notions defined before. It is the usual way to define indirect measures [1].

(b) By logical rules like the following: "The attribute A_i of the notion O has the value P when the values of the attributes of $\{A_m\}$ of the notions $\{O_j\}$ are ...". It is the way to specify rating rules (in particular, essential judgment ones) basing on measured attributes of specified notions.

Procedural knowledge is an evaluation procedure and measurement, rating, and assessment procedures. The evaluation procedure corresponds to ISO 9126 and is built in evaluation tools [11]; it defines the sequence of evaluation steps. The measurement and assessment methods are implemented and used as functions, of which descriptions written in the specification language are available to the users of the system.

2. General knowledge is separated from the description of a specific metrics framework. It is represented by four classes of 'standardized' definitions of: (1) MLMs, (2) SP metrics and defects expressed in terms of MLMs, (3) methods to summarize measures, and (4) commonly used SQ models;

Specifying a metrics framework is combining standardized notions and, maybe, additional definitions of: (1) quality factors and subfactors, (2) relevant SP metrics, and (3) their scales, and (4) interrelations among them.

The acceptance criteria are expressed in terms of selected and defined notions of the metrics framework.

3. A specific evaluation specification has operational semantics. Such a specification is transformed in a set of programs representing steps of an evaluation procedure. Here, measurement is: (1) obtaining MMs of SPs and then (2) getting values of relevant SP metrics (and defect information). Rating is computing rating levels for measured values. Assessment is the final step of the evaluation procedure; here, acceptance criteria are applied to the measurement and rating result. All measured and rated values are combined in tuples and stored in a global relational database (RDB). Besides, the RBD keeps the history of the evaluation process. This allows not only to form an evaluation report, but also to support SQ management during the development process.

5 Representation of Evaluation Specification

Here, the main components of an evaluation specification are:

(1) a metrics framework as a hierarchical breakdown of factors, subfactors, and metrics for a software [12], and

(2) an acceptance criteria definition for making decision about SQ.

In accordance with the methodology [12], both some 'classic' SQ models, like Boehm's or McCall's, and different corporate standard models may be used as the basis to select a set of SQ factors and subfactors.

The descriptions of the standardized models are elements of domain knowledge and may be written as follows.

```
SQM ISO–9126 :
   main Software–quality :{Functionality,
                           Reliability,
                           Usability,
                           Efficiency,
                           Maintainability,
                           Portability} .
```

Specifying an evaluation process is selecting already existed terms of domain knowledge or/and defining new ones in the specific context. It means that a measurement scale and rules have to be declared for any selected or defined factors and subfactors. These rules may be written in a functional or/and logical manner in terms of metrics, indirect measurement functions and other subfactors. In addition, a rating mapping may be declared for each metric. The example below is based on the example of the polarity profiling from [13], pp.48–50.

Note. It is supposed that the specification of functions 'Average', 'Nearest–integer' and 'Value–of' is obvious from the context of their use. The function 'Rating' produces a rating level value for a specific metric value according to the mapping specified. The notion of rating level is defined in [11]. Here, the attribute 'Measured–value' corresponds to the metric output.

```
metrics framework Our–Framework: SQ–model {
  main Software–quality :  {Usability,
                            Security,
                            Efficiency,
                            Correctness,
                            Reliability},
        scale interval (-3, 3, by 1),
        functional definition :
          Nearest–integer ( Average ( < Value–of ('Usability'),
                                        Value–of ('Security'),
                                        Value–of ('Efficiency'),
                                        Value–of ('Correctness'),
                                        Value–of ('Reliability') > ));
  ...
  factor Reliability : {Accuracy,
                        Error–tolerance,
                        Consistency,
                        Simplicity},
        scale interval (-3, 3, by 1),
        functional definition :
          Nearest–integer ( Average ( < Value–of ('Accuracy'),
                                        Value–of ('Error–tolerance'),
                                        Value–of ('Consistency'),
                                        Value–of ('Simplicity') > ));
   ...
  subfactor Simplicity :   {McCabe–number,
                            Ncloc,
                            module–count},
        scale interval (-3, 3, by 1),
        logical definition :
          3 : if for–all value
```

```
                 for cyc–compl (_ , value)
                   Rating ('cyc–compl', value) = 'excellent'
                 and for Ncloc(_ , compilation–unit–size)
                   module–count(_, mod–count)
                   compilation–unit–size/mod–count ≤ 30 ,
            2 : ...
    metric McCabe–number :
        scale absolute, {* a linear-independent path count *}
        relation  cyc–comp ( Program–unit–name : string,
                             Measured–value),
        rating level :
          'excellent' :    [1, 10 [,
          'good' :         [10, 11],
          'fair':          ] 11,14 [,
          'poor':          otherwise;
    ... } .  (* End of the metrics framework definition *)
```

SQ requirements demand that subfactor values have to be compared with the critical value '0' before to be summarized. In addition, there are target values (shown in the table on p. 50 [13]) for the factors.

```
acceptance–criteria : Our–Framework
  acceptance : {
    'required–degree' : if Value-of ('Usability') ≥ 2           and
                           Value-of ('Security') = 3            and
                           Value-of ('Efficiency') ≥ 2          and
                           Value-of ('Correctness') ≥ 2         and
                           Value-of ('Reliability') ≥ 2,
    'marginal–degree' : if ( Value-of ('Accuracy') ≥ 0          and
                             Value-of ('Error–tolerance') ≥ 0   and
                             Value-of ('Consistency') ≥ 0       and
                             Value-of ('Simplicity') ≥ 0)       and
                           Value-of ('Software–quality') ≥ 2          }
  rejection : {
    'fail' :               otherwise   }.
```

Here, if an actual value of any factor does not reach its marginal value, or if an actual value of any subfactor does not reach the required one, then there will be the decision on rejection in despite of the decision on acceptance or rejection, which is based on the summarized value of 'Software–quality'.

6 Representation of Metrics, Defects and Functions of Indirect Measurement

In accordence with [1, 11, 12], a *metric* may be represented as a scale and a method to determine a value of an attribute of a specific SP, i.e., as the following set: (1) name of the metric/mapping; (2) declaration of the admissible input, (3) description of the measurement scale, (4) specification of the mapping (the measurement method).

A *SP defect* may be considered as a special case of metrics [1]. Then the mapping describes the condition(-s) of presence of the defect in a SP, and, in general, it defines a bijective mapping onto the nominal scale {*yes, no*}.

According to the representation technique used, there is a relation for any metrics, and after measurement to be done, there is a set of metric values for a specific SP in the RDB (the set may be considered as the table). The specification of the metrics is written in a special-purpose specification language having some features of relational/logical, functional and algorithmic languages. The simple examples of specifications of two metrics are shown below. The MMs refered in the input descriptions are specified in Section 7.

```
defect Unstructered–use–of–goto (Uug):
  input Construction–Simple–statement (C–Ss),
  scale nominal {’yes’, ’no’},
  relation D–Uug (Program–unit–name : string,
                  ’backward–jump–by–goto’,
                  Measured–value),
  logical definition:
    ’yes’ : if there-exist C–Ss (Program–unit–name,
                                  Stat–no-1,
                                  - ,
                                  ’GOTO’,
                                  - ,
                                  Stat–no-2,
                                  ...);
                 and Stat–no-2 < Stat–no-1,
    ’no’ : otherwise .
```

```
metric Ncloc:
  input Symbol–End–of–line (S–Eol);
  scale absolute,
  relation Ncloc (Compilation–unit–name : string,
                  Measured–value),
  functional definition:
    Number–of–p-elements (Compilation–unit–name, ’S–Eol’, 2, <_, 0,_>) .
```

Note. Here, the function ’Number–of–p-elements’, The Number of Proper Elements in the List, (Compilation–unit–name, Tuple–name, No–of–the–element–which–is–the–list, < Pattern >) gets a value of how many times the pattern ’Pattern’ occurs in the list, which is the ’No–of–the–element...’th element of the tuple ’Tuple–name’ built when the program/compilation–unit ’Compilation–unit–name’ has been analyzed.

We should note that metric values may be defined not only basing on MLMs, but on other models of SPs as well, e.g., on control flow graphs. These models, in turn, are represented by relations in the same manner.

Together with the SP metrics and defects, the LOA implies representation of knowledge of the general-purpose measurement functions like 'Weighted-average' or 'Correlation'. They may be used to express indirect functional dependences among measurement scales. Specifications of the functions look like ones of metrics with the exception of defining application conditions in their scales (meaningfulness condition). It is necessary to provide the automatic check if the result is valid. A part of the specification of the function 'Weighted-average' is shown below.

```
function W-average :
  parameters (list-of-pairs-of-expressions-and-weights) :
                  < exp : real, weight : real > ),
  scale
    interval  if all exps have got values of the same interval scale,
    ratio     if all exps have got values of the same ratio scale,
  functional definition : ... .
```

7 Language and Product Measuring Models

Method of the *measuring language models* is the basis of LOA [6, 8].

The measuring model M (or M^{mea}) *of a language L* is the quadruple:

$M^{mea} = (T^{mea}, N^{mea}, P^{mea}, S^{mea}),$

T^{mea} is the set of MMs of language symbols. Any model includes a set of symbol attributes and a measurement scale for each attribute. For example (for ISO Standard Pascal [14]):

Symbol-End-of-line (S-Eol):

(1) program name [or a reference to the name] : *string*,

(2) tuple-list : < *integer*, *integer*, *integer* >:

(2-a) position in the line,

(2-b) the number of non-blank non-comment characters in the line,

(2-c) the number of comment characters in the line.

N^{mea} is the set of MMs of basic language constructions, i.e., those constructions from that all other constructions could be built. Each construction MM includes a set of construction attributes and a measurement scale for each attribute. For example (for the same language):

Construction-Simple-statement (C-Ss):

(1) compound name of the program unit in which the statement is defined [or a list of references to the names] : *string*,

(2) number of the simple statement : *integer*,

[(3) address of the statement text beginning : *integer*,]

(4) type : *string*,
(5) reference to the label declaration [or the label value]
    (if the statement is labelled) : *integer*,
(6) list of references to next executable simple statements : <*integer*>,
(7) list of references to operands of the simple statement : <*integer*>,
[(8) computer resources use].

P^{mea} is the set of rules how to form symbol and construction MMs and measure values of their attributes;

S^{mea} is 'the highest level language construction' covering semantics of all different constructions.

As a matter of fact, the MLM is a rigorous definition of a measurement tool for the language L, i.e., a definition of measurement functions of MLP and the format of the measurement result generated by MLP. In other words, MLM might be considered to be the generative grammar M^{mea} defining the language L^{mea} to represent the measurement results Rs for SPs written in the language L; $L^{mea} = L^{mea}(M^{mea})$.

The static measurement result for a program may be considered as a set of constant atomic propositions (CAP) of a relational (logical) language. Each argument of any CAP is a value of an attribute of a symbol or construction found in the program. It is the *static MM* of the SP.

The complete measurement result for a program is a join of its static MM and the measurement result for its dynamic attributes during Z its executions. The second component is the *dynamic MM* of the program obtained on the base of a specific number of its executions. It is a set of CAPs as well.

8 Application of the Approach to Automatic Evaluation

In addition to development of formal specifications in the context of the methodology [12] and the ISO Standard [11], the approach allows building software systems for automatic SP evaluation. Figure 2 shows the scheme of activities to develop such a system for a specific evaluation process.

Here, MLPs are used to obtain MMs of SPs. If a SP writing language has no operational semantics, its MLP builds the static MM only. For a language with the operational semantics, the virtual machine of its MLP generates MMs for each program execution. All together, the models of individual executions form the dynamic MM of the program. The tools like LEX and YACC may be used to develop MLPs. MLPs are built once for each SP writing language. They are general–purpose measurement tools.

Metric and defect definitions written in the specification language are compiled into a set of executable modules once too. Calls of the modules do second level measurement in terms of the relevant SP metrics in the context of a specific measurement procedure. The procedure is generated automatically by each metrics framework description.

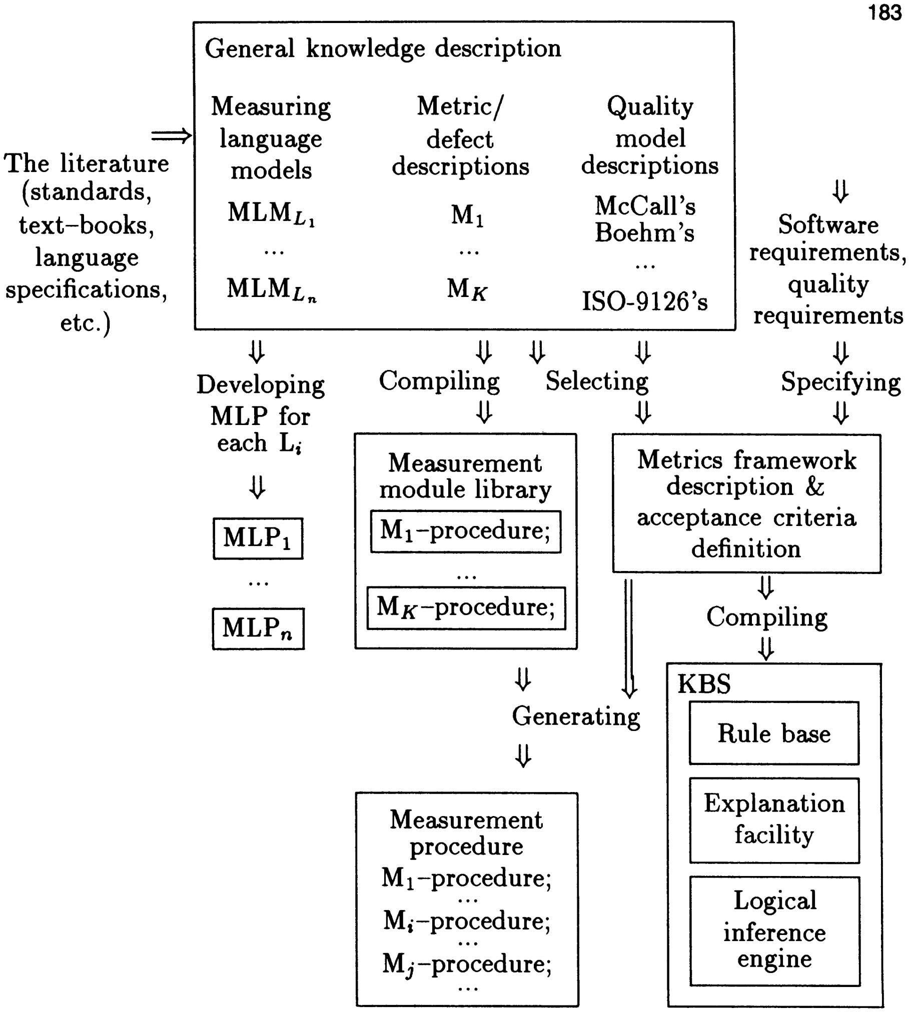

General knowledge description
Measuring language models
Metric/ defect descriptions
Quality model descriptions
MLM$_{L_1}$
M$_1$
McCall's
Boehm's
...
MLM$_{L_n}$
M$_K$
ISO-9126's
The literature (standards, text-books, language specifications, etc.)
Software requirements, quality requirements
Developing MLP for each L$_i$
Compiling
Selecting
Specifying
MLP$_1$
...
MLP$_n$
Measurement module library
M$_1$-procedure;
...
M$_K$-procedure;
Metrics framework description & acceptance criteria definition
Compiling
Generating
KBS
Rule base
Explanation facility
Logical inference engine
Measurement procedure
M$_1$-procedure;
...
M$_i$-procedure;
...
M$_j$-procedure;
...

Figure 2. The scheme of activities to develop the KBS.

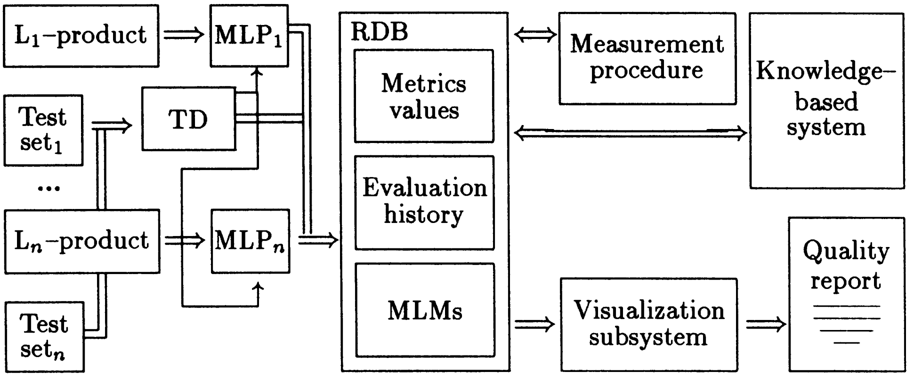


Figure 3. The scheme of knowledge-based automatic evaluation tools.

The description and the criteria definition are automatically transform into a rule base, which together with the logical language processor and built–in explanation facility constitute a KBS, the user workbench.

Figure 3 shows the structure of the system and the scheme of its use at SP evaluation. A test driver (TD) is used as the 'manager' at the first level of software measurement. A relational DBMS is used to manage the RDB, which keeps the measurement results and some intermediate evaluation information. To draw evaluation reports, the system has an imaging subsystem, which can display and print usual forms of reports.

References

1. Fenton N, Pfleeger S. Software metrics: A rigorous & practical approach. Second ed. International Tompson Computer Press, 1996
2. SCOPE Consortium, Hausen H-L, Welzel D. SCOPE Technology Work Report SC.93/009/GMD.hlh.dw/T2.1/DT/03, GMD Sankt Augustin, Germany, 1993
3. Cogan B. Engineering expert systems for software quality assessment. In: Lum E (ed) Proc. 2d Int. Comp. Science Conf. on Data and Knowledge Engineering: Theory and Applications, Hong Kong, 1992, pp.349-357
4. Cogan B. Intelligent automatic software engineering tools for software studying and assessment. In: Ross M (ed) Proc. 3d Int. Conf on Software Quality Management, Seville, Spain, 1995, Computational Mechanics Publication, Southampton, 1995
5. Cogan B, Hunter R. Language–oriented approach to software measurement. In: Bray M, Ross M, Staples G (eds) Proc. Software Metrics Symposium, Berlin, Germany, IEEE Comp. Society Press, 1996, pp.3-8
6. Cogan B, Matveeva T. A relational approach to software measurement and quality assessment. In: Cowderoy A (ed) Proc. 7th European Software Control and Metrics Conference, Wilmslow, UK, 1996, pp.280-291
7. Cogan B, Hunter R. Definition and collection of metrics for comprehensive software measurement. Software Quality Journal 1996; 5; 211-220
8. Cogan B, Ostrouchova S. Building a measuring model of a programming language. Research Report RR/96/200. Glasgow: University of Strathclyde, Department of Computer Science, 1996
9. SCOPE Consortium, Hausen H-L, Welzel D. Specification of software evaluation and certification - Formal model. SCOPE Report SC.93/019/GMD.hlh.dw/T21/RP/02, GMD Sankt Augustin, Germany, 1993
10. Buchanan B, Bobrow D, Davis R, et al. Knowledge–based systems. Annu. Rev. Comput.Sci., 1989-1990, 4, pp. 395-416, Palo Alto, Calif., 1990
11. ISO/IEC 9126, Software product evaluation - Quality characteristics and guidelines for their use, 1991
12. IEEE standard for a software quality metrics methodology: ANSI/IEEE Std 1061-1992, 1995
13. Watts R. Measuring software quality. NCC Publications, The National Computing Centre Limited, Manchester, England, 1987
14. Jensen K, Wirth N. Pascal User Manual and Report. Springer-Verlag, 1985

Australian Software Developers Embrace QA Certification

Aileen P Cater-Steel and Edmond P Fitzgerald
Department of Information Systems, Faculty of Business
The University of Southern Queensland
Toowoomba, Australia

Abstract

This paper details a research project undertaken to assess the extent of adoption of quality assurance (QA) certification by Australian software developers. A brief history of government QA policy, the catalyst in the sudden interest in certification, is included. Primary data for the study were gathered from a survey of 1,000 Australian software developers, and were used to determine the extent of adoption of QA certification by Australian developers, their organisational characteristics, capability maturity and perceptions regarding the value of QA certification. Secondary data from the JAS-ANZ register of certified organisations enabled validation of survey responses and extrapolation of QA certification adoption.

Major findings of the study revealed that 11 percent of respondents are certified to ISO 9001 or AS 3563, seven percent are in progress and 21 percent plan to adopt QA certification. It also revealed that specialist developers are adopting QA certification at twice the rate of in-house developers. Other factors found to be associated with adoption of QA certification are large development groups, developers with government or overseas clients, organisations with whole- or part-foreign ownership, and organisations undertaking corporate TQM initiatives. From the findings, detailed implications are drawn for managers and policy analysts.

1 Introduction

With the increasing functionality and decreasing cost of computer hardware, there is a significant trend for organisations to implement new computer systems, and to redevelop legacy systems [1]. As well as being critical to the operation of their business, more software is becoming embedded in products and services [2]. Despite the growing importance of software, most software projects are completed over schedule and over budget [3]. For example, a survey conducted by the United Kingdom's Department of Trade and Industries found that 66 percent of all software projects were completed later than planned; 55 percent of projects ran over budget; and 58 percent had major problems [4]. That the Australian software industry is not exempt from similar problems is evidenced firstly by some projects being cancelled, often after a significant investment, for example the Commonwealth Bank's $100 million Mainstream project [5]; and secondly by the delivered product sometimes being infested with bugs, or falling short of the

C. Hawkins et al. (eds.), *Software Quality Management VI*

expectation of the client, for example Telstra has spent $300 million on its Flexcab core billing system and '... all the code will now have to be re-examined, changed where necessary and retested from scratch for year 2000 compliance' [6].

These continuing problems constitute what has been referred to as the 'software crisis' [7]. Thus for large, complex software projects, it is important that the purchasers be able to assess the capability of would-be developers. However, as professional qualifications are not required to practise as a software developer, a potential client has no formal way of judging the competence of would-be developers. The use of third party certification may help to alleviate this problem. As well as describing changes in government QA policy this paper provides details related to the extent of current and future adoption of QA certification by Australian developers.

2 Australian Government Policy on QA Standards

Standards are profoundly important to industry and form a critical element in the future success of Australian industry, domestically and internationally [8]. Government QA purchasing policy is cited in the press as a key motivator for many organisations deciding to adopt QA certification [9, 10].

In 1987, during the Hawke/Keating government, Senator John Button commissioned a committee under the chairmanship of Dr Kevin Foley to prepare 'The Standards, Accreditation, Quality Control and Assurance Report'. This report formed the basis of the Federal Government's Quality Assurance Policy which was announced in May 1992. Departments and agencies were entitled to ask suppliers for quality assurance for the supply of goods and services from 1 July 1993 for manufactured goods and related services, and from 1 January 1994 for the provision of 'unrelated' services [11]. Some state governments had already adopted ISO 9000 as a mandatory requirement for all suppliers of goods and services: Queensland (January 1990), South Australia (January 1991), and Western Australia (August 1991) [12]. Although the federal government did not formally adopt a policy of QA certification for goods and services, many government purchasers stipulated on tenders that parts or all of ISO 9000 was required. The effect was sensational:

> ... soon Australia was proudly leading the world in the take-up of ISO 9000. By the start of January 1993, Australia and New Zealand had nearly 7% of all the certificates issued worldwide (the US had 4.3%). By December 1995, this had grown to 8.3%, with almost 9000 certificates issued [13].

But despite the rush to certification, many small businesses were appalled at the overhead in terms of cost and bureaucratic procedures, and they lobbied for relaxation in relation to government purchasing requirements. In response to this backlash, John Sprouster, CEO of the Australian Quality Council, announced in 1994 that ISO 9000 was inappropriate and too costly for small business. Prime Minister Paul Keating commissioned Bruce Kean to lead the Committee of Inquiry into Australia's Standards and Conformance Infrastructure, releasing the 'Linking Industry Globally' [8] report. This committee found that ISO 9000 should only be compulsory in the international marketplace [14]. However, before Keating could act on this report, his party lost power to the Howard-led coalition in the March 1996 Federal election. Geoff Prosser, Small Business Minister in the newly-elected Howard government, announced in July 1996 that the Government was turning its back on ISO 9000 [15, 16] .

Many of the state governments are also reviewing their policies:

- Queensland had a mandatory policy in place for ISO 9000 from 1992 to October 1996, but now QA is mandatory only for high risk projects;
- Victoria is introducing a three-tier system;
- Western Australia is reviewing its mandatory policy;
- New South Wales and South Australia are not carrying out any reviews (both have a system whereby different government agencies require varying degrees of ISO 9000, depending on risk and value) [15, 16].

3 Standards Applicable to Australian Developers

Due to its generic nature, ISO 9001 is difficult to interpret in the context of software development, so guidelines have been produced to help software developers and auditors apply it to the software industry.

In the United Kingdom, a guide was produced by the British Computer Society: 'TickIT: Guide to Software Quality Management System Construction'. Based on ISO 9001, it melds the guidance of ISO 9000-3 (Guidelines for the application of ISO 9001 to the development, supply and maintenance of software) with the requirements of ISO 9001 and contains five sections: introduction, the application of ISO 9001 to software, a purchaser's guide, a supplier's guide, and an auditor's guide [18, 19].

Australia has also developed and promoted its own standard, AS 3563 'Software Quality Management System' through the efforts of the QR/3 Committee of Standards Australia. This certificatable standard extends ISO 9001 into project planning, requirements specification, and the development of programming and documents. The AS 3653 standard is highly regarded in international circles and was adopted as standard 1298 by the International Electrical and Electronic Engineers (IEEE) in 1992 [20, 21].

At the time of the data collection for this study (May 1995), Australian software developers seeking quality system certification had a confusing situation with three different options for certification: AS 3563.1, ISO 9001 (as applied via AS/NZS 9000.3), or joint certification to AS 3563 and AS/NZS 9001. In customising AS 3563 for software development, a new clause 'Control of development environment' has been included which explicitly covers sub-contractor assessment, programming standards, maintenance and configuration management.

The standards are continually reviewed and updated as a result of work conducted by standards committees and technical working groups. For example, committee QR/3 (responsible for AS 3563 and now including representatives from New Zealand) has developed a new 1996 guideline, AS/NZS 3905.8: 'Quality systems guidelines Part 8: guide to AS/NZS ISO 9001:1994 for the software industry'. AS 3563 has now been declared 'obsolete' by Standards Australia. This means that software developers will be seeking certification only to AS/NZS 9001 in the future, using the AS 3905.8 guidelines [22]. One of the major enhancements included in AS 3905.8 is the inclusion of examples of statements, documents and plans specific to software development, which practitioners should find very helpful.

4 Research Design

A survey of QA adoption by Australian software developers was conducted by Gori in 1992. His sample was drawn from a list of the largest organisations in Australia; it included large and small in-house developers but only large specialist developers. However, 97 percent of the businesses which make up the computer service industry employ less than 20 people [23]. Hence most specialist developers, being small businesses, were not included in Gori's sample.

4.1 Research Questions

The purpose of the study was to investigate the extent of adoption of QA certification, organisational characteristics of adopters, the capability maturity of Australian software developers, and their perceptions regarding certification costs and benefits.

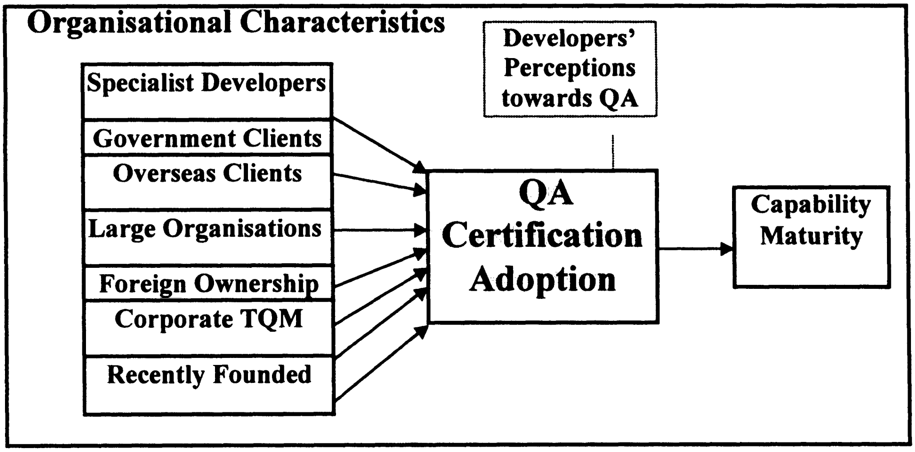


Figure 1 Research Model

As depicted in figure 1, the study addressed the following four research questions,:

(i) What is the extent of adoption of third-party certified QA standards by the Australian software development industry?

(ii) Do QA certified developers exhibit common organisational characteristics (external clients (specialist developers), government clients, overseas clients, organisational size, foreign ownership, corporate TQM, recently founded)?

(iii) Is higher capability maturity associated with adoption of QA certification?

(iv) How do developers perceive the value and effects of QA certification in relation to its costs and benefits?

As it is not possible to provide a full report on the findings within the limitations of this paper, the focus will be on reporting the extent of adoption, summarising organisational characteristics associated with QA certification adoption, and interpreting the findings in terms of implications for managers and policy analysts.

4.2 Survey Design

A questionnaire was designed and then pilot tested in two stages with suggested modifications from the first stage incorporated for the second stage of testing. The questionnaire included 12 demographic questions, 5 questions relating to QA certification progress, 13 statements probing developers perceptions regarding the value of QA certification and government QA policy and 33 questions (based on the SEI maturity questionnaire) to determine software engineering practices.

4.3 Sample Selection

The unit of analysis was Australian organisations undertaking software development. The target population was all organisations in Australia which develop software for sale (specialist developers) or for their own use (in-house developers). Two sampling frames were used, as a single list containing both types of developers was not available. Firstly, all specialist Australian software developers were extracted from the 'Oz on Disc' Yellow Pages Database. From the total population of approximately 4,000 software developers, a random sample of 500 was selected. To ensure in-house developers were adequately represented in the study, a random sample was drawn from the MIS 3001 database which contains details of the 3,500 largest users of IT in Australia and New Zealand. To maximise the probability that the selected organisations undertake software development, organisations which had not indicated usage of CASE or 4GL tools were eliminated. From the remaining 1,690 records, 500 Australian organisations were then selected at random.

5 Extent of Adoption of QA Certification

5.1 Current Status

The primary data from the survey, shown in table 1 revealed that 11 percent of respondents are certified to ISO 9001 or AS 3563, seven percent are in progress and 21 percent plan to adopt QA certification. It also revealed that specialist developers are adopting QA certification at twice the rate of in-house developers.

STATUS ISO 9001 or AS 3563	DEVELOPER TYPE					
	In-house (n=165)		Specialist (n=123)		Total (n=288)	
	Freq	%	Freq	%	Freq	%
No Plans	121	73.3	57	46.3	178	61.8
Planned	25	15.2	34	27.6	59	20.5
In Progress	8	4.8	12	9.8	20	6.9
Certified	11	6.7	20	16.3	31	10.8

Table 1 Comparison of certification status between in-house and specialist developers

Analysis of information provided from the JAS-ANZ register [24] shows that the number of organisations holding certification for software development has increased each year from 1991 to 1996. The number of certified organisations rose 78 percent during 1994 and 44 percent during 1995. At the time of the survey (May 1996), accredited registrants had already advised JAS-ANZ of a further 21 certificates for ISO 9001 or AS 3563 for 1996, bringing the number of organisations certified for software to 120.

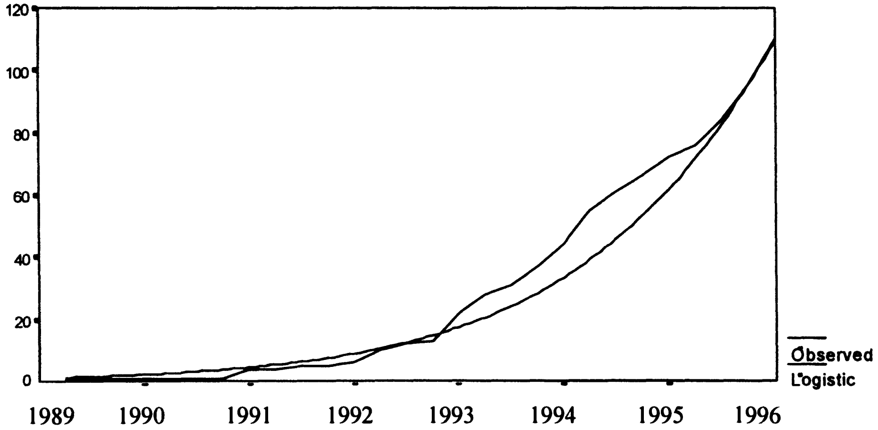


Figure 2 Logistic growth curve - cumulative certifications 1st qtr 1989 to 1st qtr 1996

The growth logistic curve [25] is often used to show how something grows or increases with time. When SPSS is applied to fit the logistic model to the secondary data with an estimated upper bound of 500 certifications, the estimated equation is Y=1/(1/500+2.1403*.8115**t). As can be seen from figure 2, this model closely fits the data and is confirmed by a coefficient of determination (.965) which indicates that 96.5 percent of the variation is accounted for by the time variable (Beta=.374409, p<.001, adjusted R^2=.96383, n=28).

Figure 3 shows that Queensland (Australia's third most populous state) has the highest proportion of certified respondents with 17 percent of respondents certified to ISO 9001 or AS 3563. This may be explained by the fact that Queensland was the first state in Australia to adopt minimum quality standards in its purchasing policy. Figure 3 also illustrates that the states of South Australia, Tasmania and the Northern Territory do not have significant QA certification activity, however, the low number of responses from Tasmania and the Northern Territory make it difficult to generalise.

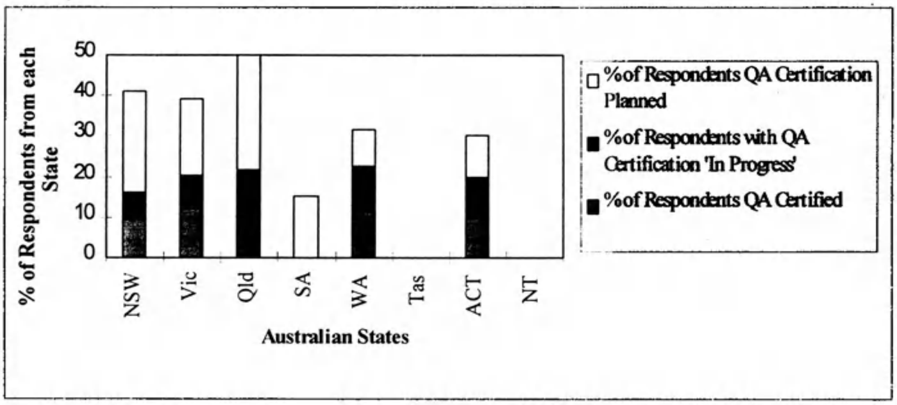


Figure 3 Certification progress shown for proportions of respondents by state

5.2 Projected Adoption

The information derived from the JAS-ANZ register can be extended by including the proportions of this study's respondents who reported that they are planning to achieve certification, or are already in the process of adopting certification. Based on the 'typical' timeframe for certification [18], it can be predicted that all the organisations currently with QA certification in progress will achieve their certification goal within 12 months, and those that are in the planning stage will have completed the process within two years. The survey responses indicate that the ratio of certified: in progress: planned is 10.8 : 6.9 : 20.5. Applying these proportions and extrapolating adoption, based on the May 1996 figures of the JAS-ANZ register, in 12 months there would be an additional 70 organisations certified, and in a further 12 months (i.e. May 1998), an additional 209, giving a total of 389 organisations certified for software development.

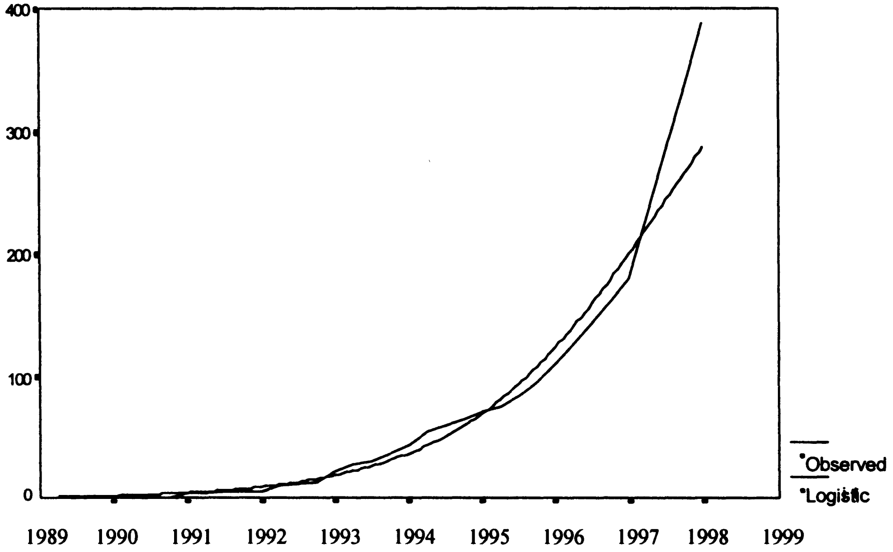


Figure 4 Current/ predicted adoption of QA certification by developers, 1989 to 1999

The logistic growth curve proposed above fits the predicted certification activity (refer to figure 4). However it would appear that the predicted upper bound of 500 certificates of the proposed model is too conservative.

It is difficult to compare the extent of adoption with previous Australian results as the only similar large scale survey, that conducted by Gori in 1994, sampled only large organisations, sourced from the MIS 3000 database. Gori [20] found that 14 percent of his respondents had no plans for a quality management system (QMS), and that many organisations who were planning or preparing their QMS did not intend to seek third party certification. These results are consistent with the findings in table 5.21, which shows that 73 percent of in-house developers (sampled from the MIS 3000 database), have no plans to achieve QA certification.

5.3 International Comparisons

The Hong Kong survey [26] sampled only *specialist* developers. Our survey data indicated that the QA activity of specialist developers in Australia is similar to that found in Hong Kong, where around half of the Independent Software Vendors (ISVs) there intend to, or are already implementing ISO 9000 [26]. The overall proportion of certified organisations in Australia (almost 11 percent) appears to be much less than in the United Kingdom. Davis, Thompson, Smith and Gillies [27] found that 22 percent of their 151 respondents were third party assessed, and that almost half of these certificates had been recently issued.

5.4 Effect of Organisational Characteristics

In order to understand how organisations are influenced towards adoption of QA certification, data on organisational characteristics were collected and analysed. Support was found for the following hypotheses:

- specialist developers are more likely than in-house developers to adopt QA certification;
- developers with government-funded clients are more likely than developers without government-funded clients to adopt QA certification;
- developers with overseas clients are more likely than developers without overseas clients to adopt QA certification;
- large specialist organisations are more likely than small specialist organisations to adopt QA certification;
- organisations funded by overseas capital are more likely than wholly-Australian-owned organisations to adopt QA certification;
- organisations involved in corporate TQM are more likely than non-TQM organisations to adopt QA certification.

6 Implications for Practice and Policy

In this section, the implications of our study's findings for private sector managers, government policy analysts and public sector managers are examined.

6.1 Private Sector Managers

The extent of adoption of QA standards uncovered by this study, has important implications for managers of specialist and in-house software development groups. There are also implications for managers in organisations which are clients of software developers, referred to here as 'client managers'. Considering the actual and forecasted number of adoptions, the following recommendations are based on the literature and the findings of this study.

6.2 Software Development Managers

- The extent of adoption by specialist developers revealed by this study indicates that those without QA certification may soon need it as a competitive necessity. This implies that the early adopters will need to continuously improve as the competitive advantage they currently enjoy is unlikely to be sustainable. To protect their market, it is recommended that specialist developers determine if their clients (current and potential) are planning to adopt QA supplier certification.

- It is further recommended that small newly founded software developers consider implementing QA standards to overcome barriers identified by Ritter [28] such as lack of reputation and lack of attractiveness as business partners.

- With the recent trend towards outsourcing IS services, in-house developers may find that QA certification provides a defence against outsourcing, and may protect them against an outsource (certified) bid. Therefore, it is recommended that in-house groups determine if QA certification is being adopted by their parent organisation's customers or competitors, and actively participate in any corporate QA activity.

- To ensure that QA certification can be adopted and maintained in-house, minimising expensive external consultants, it is recommended that organisations consider QA training for existing staff, and specify QA skills in recruitment criteria.

- Finally, to facilitate adoption of QA certification, it is recommended that software development groups evaluate the use of more advanced CASE tools, groupware and document management systems.

6.3 'Client Managers'

By providing information about the number and organisational characteristics of certified software developers, this research has produced a profile of software developers in different stages of QA certification which 'client managers' may find very informative in deciding whether to insist on QA certification of their developers. In the past, insistence on QA certified developers for software contracts has limited the pool of potential suppliers. This research shows that the pool is growing at a significant rate, thus facilitating use of certified developers.

Although 'client managers' in Australia have not been surveyed regarding their QA policy or perceptions regarding the value of QA certification, a number of recommendations are made based on the findings from this study. The recommendations may assist these managers in deciding whether to require QA certification or not.

Therefore, it is recommended that:

- as higher capability maturity is associated with QA certification [29], and as the number of QA certified developers is increasing, clients should consider QA certification as a pre-requisite in evaluating potential developers;

- in view of the concerns highlighted in this study regarding the value of QA certification as a reliable indicator, client managers consider other indicators as well as certification in assessing potential developers;

- client managers consider the risks and value associated with a project before insisting on QA certification; for low-risk, low-value projects the added cost of QA certification may not be justified.

6.4 Public Sector Policy Analysts and Managers

Although this study did not seek to determine cause and effect relationships (associations only), it appears that government policy has a major influence on the adoption of QA certification by software developers. It was found that 23 of the 31 certified organisations have government-funded clients. Also, as shown in figure 3, Queensland - the first state to adopt mandatory QA standards - provided the highest proportion of certified responses (17 percent of Queensland respondents are certified).

Although government policy has been successful in terms of increasing the adoption of QA certification by suppliers, it appears to have failed in its objective to boost the competitiveness of local industry. The findings from this study support the view presented in the media [10] and by other research [30, 9, 31, 16] that government policy has benefited large foreign owned organisations at the expense of small local developers.

Even the architect of the Federal Government's QA policy, Dr Kevin Foley, now admits that '... in retrospect the quality movement did not take into account the cost to business and the importance of the risk and value of the contract' [13]. Another problem highlighted in the media [15] involves the implementation of the policy: 'Government purchasers publicly claimed that there had never been a policy of mandatory ISO, and they would retrain staff to stop insisting on it'. Roger Dewar, Principal Project Officer with Queensland Purchasing (Department of Public Works and Housing) explained that government purchasing officers are generally risk averse, and tend to over-specify QA in contracts [17]. He went on to explain that training purchasing officers about changes to government QA policy is effective, but changing the habits of other 'officers who purchase' is difficult. As well as complaining about QA compliance being over-specified, practitioners also complain that on occasions, the stated QA policy was ignored, with contracts being awarded to non-QA certified organisations who submitted the lowest quote [32]. In relation to government regulatory bodies setting standards, these problems confirm David and Greenstein's [33] comment that 'designing efficacious actions and appropriate guiding principles for every situation opens a large research agenda'.

7 Conclusion

With half the specialist respondents involved in QA certification and the dramatic rise in certificates issued to specialist and in-house developers, the adoption of QA represents the most pervasive effort to date by Australian software developers to improve their software processes. However, changes to government policy, although well-intentioned, may result in a competitive imbalance biased against local developers.

References

1. Fried, L. 1995, Managing Information Technology in Turbulent Times, Wiley-QED, NY.
2. Senn, J.A. 1995, Information Technology in Business: Principles, Practices, and Opportunities, Prentice Hall, NJ.
3. Ashton, S. 1996, The Value of User Evaluation, ComputerWorld, 27 Sep, pp. 44-5.
4. Russell, J.F. 1993, ISO 9000 Tackles Software Development, Electronic Business Buyer, Oct, pp. 122-3.
5. Maiden, M. 1996, Revealed: How CBA Fell into Techno-Trap, The Sydney Morning Herald, 3 May, pp. 21-2.
6. Birmingham, A. 1996,Telstra Time Bomb Ticks to $100m, ComputerWorld, 21 Jun, p. 1.
7. Tan, M. & Yap, C.Y. 1994, A Study of TQM Practice on Software Development in Singapore, TC8AUS IFIP I S International Working Conference, May, pp. 575-84.
8. Committee of Inquiry into Australia's Standards and Conformance Infrastructure. 1995, Linking industry globally: Overview, Standards and Conformance Infrastructure Inquiry, Australian Government Publishing Service, Canberra.
9. Gome, A. 1995, ISO 9000 Inappropriate for Small Business, Business Review Weekly, 13 Mar , p. 52.
10. Financial Review 15 Jul. 1996, editorial.
11. AIIA 1993, Quality Management Issues: An IT Perspective, AIIA, Deakin, ACT.
12. Wilson, D. N. 1993, Software Quality Assurance in Australia, SQM, Eds M.Ross et al., Comp. Mechanics Pub., Southampton Boston, pp 911-24.
13. Gome, 1996, Total Quality Madness, Business Review Weekly, 30 Sep, pp. 38-44.
14. Hudson, T. 1995, The Quality Question, Ausindustry, July/August, pp. 14-5.
15. Gome, A. 1995, Canberra May Become Less Gung-ho on ISO, Business Review Weekly, 4 Sept , p. 68.
16. Dwyer, M. 1996, Canberra Takes Action on Standards, Aust Financial Review, 15 Jul, pp. 1,4,16.
17. Dewar, R. 1997, pers. comm., 11 March.
18. Kantner, R. 1994, The ISO 9000 Answer Book, Wiley, New York.
19. Smith, D.J. 1995, Achieving Quality Software (3rd edition), Chapman & Hall, London.
20. Gori, G. 1994, The State of Quality Assurance in the Software Industry in Australia, MPM Graduate Project Report, UTS, Sydney.
21. Jenner, M.G. 1995, Software Quality Management and ISO 9001, Wiley & Sons, N.Y.
22. Johnstone, G. 1996, pers. comm., 29 October.
23. Madden, R. 1995, Computing Services Industry Australia 1992-93, ABS ACT.
24. JAS-ANZ 1996, pers. comm., 19 August.
25. Montgomery, D.C. & Peck, E.A. 1992, Intro to Linear Regression Analysis, Wiley NY.
26. HKPC & IDC 1995, Vol2: Phase 1 Study - Industry Analysis, in Consultancy Study on Hong Kong's Software Industry 1994-95.
27. Davis, C.J., Thompson, J.B., Smith, P., & Gillies, A. 1992, A Survey of Approaches to Software Quality within the United Kingdom, Occasional Paper, University of Sunderland, vol. 92, no. 5, pp. 1-74.
28. Ritter, T. 1997, Network Competence: A Framework for Effectiveness in Technological Networks, in Doctorial Consortium on Interfirm Relations and Networks.
29. Cater-Steel, A.P. & Fitzgerald, E.P. 1997, Quality Assurance Certification: Adoption by Australian Software Developers and its Association with Capability Maturity, Proc APSEC & ICSC, Hong Kong pp. 11-23.
30. Zampetakis, H. 1994, Investment in Quality Accreditation Queried, Aust Financial Review, 31 Jan , p. 28.
31. Kennedy, A. 1995, Counting the cost of ISO accreditation, Business Review Weekly, 7 July , pp. 48-9.
32. Dore P., 1997 pers. comm. 11 March.
33. David, P.A. & Greenstein, S. 1990, The Economics of Compatibility Standards: An Introduction to Recent Research, Economics of Innovation & New Technology, pp.3-41.

Management Systems: the End of the Beginning?

Roderick H Macmillan MISTC
British Telecom Labs, Martlesham Heath
Ipswich, UK

Abstract

For many years quality managers have been striving to merge the requirements of management systems into "business as usual". Traditional paper-based formats have made this very difficult to achieve. However, the advance of technology and the transfer of information to the electronic medium have provided new opportunities. In the new medium the EFQM Business Excellence Model may be used as a navigational tool to finally solve this problem.

Introduction

Management systems have come a long way over the years, and are now changing at an accelerating rate. Advances in technology are providing the means to solve several of the fundamental problems with which quality managers have grappled for years. It is now possible to correct the mistaken perception that a quality management system is a bolt-on addition to "real work". The end of the beginning may have arrived.

Techniques for delivering up-to-date information in an accessible form to everyone in an organisation are now established, although many organisations have yet to make the change. However, it is not only the speed and medium of delivery that the new technology has changed. Fundamental changes in the way information is structured and used are now demanded.

At BT Laboratories Network Engineering Centres, part of the national research and development centre of British Telecommunications plc, in Suffolk, a paper-based management system filled several A4 ring-binders for a number of years. The documents were not particularly easy to use or maintain, and were eventually converted into electronic hypertext form [1] [2] using *Microsoft Multimedia Viewer*. This improved accessibility and hence adherence to requirements. The subsequent introduction of an intranet brought about further significant changes. Accessibility was again a key consideration, and numerous navigational techniques were used. An alphabetic index proved most popular, although a

C. Hawkins et al. (eds.), *Software Quality Management VI*

dedicated search engine was also well used. Other navigational tools such as business processes, ISO 9001 and topic groups found a lower level of application. Although accessibility had been improved immeasurably, and problems such as issue and change control had been solved, one basic problem remained. The management system was still regarded as a separate entity – a self-contained electronic book addressing particular processes. Everyday work was, in the eyes of a significant number of users, something quite different.

History

The way in which an organisation perceives its management system is very important. A system that is seen as a set of dusty volumes to which reference is only made prior to visits by third-party auditors, or in desperation, has failed. It is the task of an organisation's senior managers, and of a quality manager, to get the requirements of a management system accepted as the normal way of doing business.

One of the tools that may be used to influence the perception of users is the presentation of the management system itself. Users typically make reference to a management system when specific questions arise. Easily found, concise yet complete answers inevitably improve perception of the management system. But how can this be achieved?

Management systems are normally home grown, and early versions may be over-prescriptive, unreadable volumes that soon fall from favour. Authors love to see their words in print, and the effort involved in changing or rewriting a management system may delay for years the inevitable fresh start. It is still the case that most management systems are paper-based, and updated by a laborious reissue of volumes, documents or pages. Basic systems are patched and amended so many times by so many authors and editors that few people know the whole story.

Such a system has several major defects – any one of which may prove fatal. The information may be inaccessible and difficult to read, may be out of date, and may drift away from being relevant to an organisation's everyday activities. The obvious consequence is that it falls from regular use. But there is a more serious problem – in paper form a management system is unlikely to be fully integrated into the everyday work of an organisation. It will remain something separate from the main stream. In electronic form the situation might seem to be the same, but the new medium is more flexible and offers new possibilities.

Technology and World-wide Web

The last five years have seen considerable movement toward electronic information systems. Management systems have been put on line in a variety of ways, from word-processor files dumped onto local networks, to hyper-linked help-files and company intranets. However, it is the introduction of the intranet that has really transformed the situation. For the first time, given the correct structure, users are able to find the information they require. A good system will answer users' questions within three clicks of a mouse – and the answer will consist of no more than a few relevant sentences.

The arrival of the world-wide web, and the introduction of company intranets, has opened the final door. A single source of information can now be made available to everyone, updating the information is comparatively easy, and costs have fallen. However, the potential of a company intranet extends far beyond the management system. Managers will see applications in all sorts of areas – internal telephone directories, planning, purchasing, financial etc. Once an intranet is up and running, a myriad of additional unforeseen applications will be transferred to it. Indeed, in the space of a few short years an organisation may be totally dependent on an intranet.

The beauty of this situation is that all the information an organisation makes available to its people is brought together in one medium, possibly in a single format. Hypertext links can be used not only to link parts of a management system, but also to bring together other information. The boundaries between management system and "real work" then begin to blur. Given the careful management that this new environment demands, the potential in enormous.

The Business Excellence Model

Navigation through the maze of information that is a mature company intranet is a subject in its own right, and a great deal of effort continues to be invested in addressing the problems. Accessibility remains a key requirement, and this is most effectively achieved by using a variety of parallel techniques.

Three navigation tools have emerged as front-runners. The search engine, various alphabetic approaches, and the European Foundation for Quality Management (EFQM) Business Excellence Model (BEM) - figure 1. The BEM [3] offers major advantages because it has the potential to merge previously discrete systems with other business information.

The BEM may be used to describe any organisation. The nine top-level criteria of the model are easy to understand and applicable at any level within an organisation. Thus, the adoption of this widely accepted model as a top-level intranet page allows business information of all types to be drawn together in a convenient way. The leadership criterion, for example, points to a number of lower-level topics such as values and expectations, role models, training, priorities and communication. Management system intranet pages on these subjects are now listed under these headings alongside other related information. Management system pages are identified only by the name of an author responsible for their maintenance. From the point of view of a user, the boundaries have vanished.

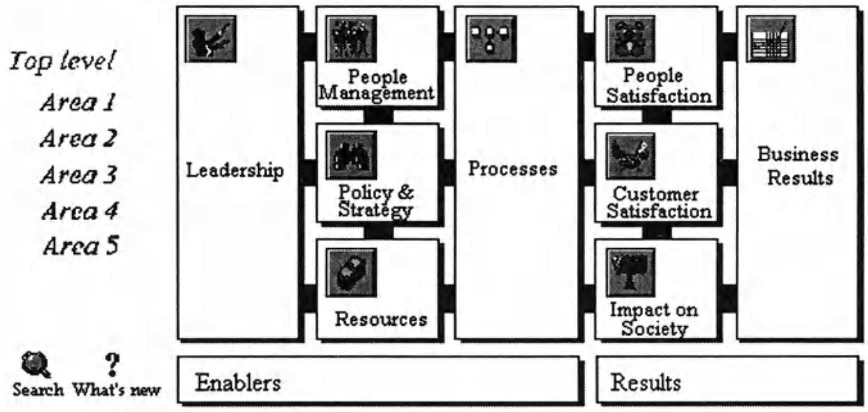


Figure 1: EFQM Business Excellence Model

Lower-level Areas

The BEM shown in figure 1 is used in a similar manner for both the top level and lower-level subdivisions of the organisation. The area in which the model is to be used may be selected by clicking on one of the areas listed to the left of the BEM. The selected area turns red. No other visible change is made to the model as areas are changed, but the map file lying behind the BEM is replaced, providing the user with links at the appropriate level.

Users in a particular subdivision of an organisation are likely to need access to top-level information supported by more detailed local material relevant to their own area. To avoid the need to change repeatedly between levels, users selecting one of the lower-level areas are presented with intranet pages divided into at least two frames. One provides hyper-links to relevant information at top level, and the other lists links to related local material. This serves as a reminder to users who are focussed on local requirements to also consider top-level material.

Search Engines

All the realisations of the BEM, at top level and local level, are supported by appropriate specifically-scoped search engines. A user who selects the BEM at top level will therefore be presented with a search engine that will search top-level information only. Local search engines are used to support the relevant lower-level realisation of the BEM. Experience will show whether it is necessary to make the top-level search engine available at all local levels.

What's New?

Changes made to information linked to the various levels of BEM are notified to users via level-by-level "What's new" sites. These are organised alphabetically, presenting users with all the changes made to topics listed under a single letter of the alphabet on a single page. A browse facility allows users to scan the entire alphabet in a couple of minutes. Summaries of significant changes made to management system requirements are also e-mailed to all users.

Other Navigation Tools

Navigation around a complicated site is best achieved by using several parallel techniques. The BEM was not designed with web-site navigation as its primary purpose, but has now shown itself to be a flexible and very useful tool. However, it is important that it is supported by other navigational techniques, such as search engines, alphabetic indexes and business processes. When used together, these techniques allow complete integration of management systems into normal business.

References

[1] Macmillan R H, *E.Quality@BT...*, *Quality Yearbook 1997/1998*, Prentice Hall, Englewood Cliffs, New Jersey, pp 4.12-4.14

[2] Macmillan R H, *Aardvark to Zulu*, Proceedings of the Fifth Conference on Software Quality Management, British Computer Society, 1997, pp285-290

[3] *Self-Assessment Gudelines for Companies*, European Foundation for Quality Management, Brussels Representative Office, Avenue des Pléiades 15, 1200 Brussels Belgium, 1997

Can Quality Survive Amidst Adversity?

W. Pat Burgess
Burgess Consulting, Inc.
St. Louis, MO USA

Abstract

Executive management of a nation wide warehousing and distribution organisation made a strategic decision to migrate from a mainframe environment onto a newer, server based platform. The underlying premise for this decision was that this platform would enable them to function more cost effectively and enable them to better react to market forces. The decision supported previous decisions to embark on a quality program and to empower employees. This program differed from previous programs in that the push was away from the Information Systems (IS) department rather than toward the IS department.

Immediately following this directional announcement, the IS staff became panicked. Within eight months of the initial announcement by management, the IS department shrank from forty-two technicians and support staff to a total of six. At first glance, this organisation which was heavily supported by technology, seemed to have been delivered a devastating blow. In reality, quite the opposite was true. This paper is a discussion of the progress of this ongoing process and the role Software Quality Management has played in rebuilding the IS department.

1 Background

In door to door distributions there are two methods of distribution: blanket and segmented. In a blanket distribution, every location within a geographical area receives the same item(s). In a segmented distribution, specific locations within a geographical area receive different item(s) based on demographic information. In addition to delivery locations, there may also be delivery instructions as to how an item is to be delivered or the placement of an item at a specific location.

Our client (referred to as D2D) is a warehousing and distribution organisation that warehouses and delivers various products door to door. It receives delivery requirements from its customer and creates subsets of delivery addresses, or locations, optimised based on like items, the number

C. Hawkins et al. (eds.), *Software Quality Management VI*

of locations to be delivered, the size of the items being delivered and the quantity being delivered to each location. Delivery optimisation and the printing of delivery instructions had previously been done on a mainframe computer. Over the previous twenty-five years, as processes were automated, the IS department had evolved into the largest department within the organisation.

Although recognised as the quality leader in their industry, D2D was loosing more and more contracts to the lower bids. This continuous market pressure to reduce costs caused them to review their most basic business processes. This evaluation gave rise to their quality initiative. After the initial training and a few team-building meetings, the program lost momentum and stalled. However, continuous market pressures to reduce costs once again caused executive management to re-evaluate their direction. Based on a third party feasibility study, management once again decided to attempt to reduce operating costs through automation: specifically to migrate from their mainframe computer environment onto a more economical server based platform. It was their belief that more powerful PC based tools would provide the ability to respond more quickly to the changing market and the growing popularity of segmented deliveries. It had been determined that with a few modifications, systems developed for maintaining delivery information in the field could perform many of their critical processes on a server based platform.

2 Migration

There is a management theory of thirds that states: for any corporate decision, one third of those affected will disagree with the decision; one third will agree; and one third will be indifferent towards it. The uncommitted third is the crucial group to win over. This concept became very apparent immediately following the directional announcement by management. Within three months, the department director had resigned, and within eight months of the announcement, the IS department shrank from forty-two technicians and support staff to a total of six employees. Two supervisors, two programmers, one computer operator and one clerk were all that remained. At first glance, this organisation which was heavily supported by technology, seemed to have been delivered a devastating blow. In reality quite the opposite was true.

As the migration progressed, it became evident that the new platform would indeed carry the load. It became a rallying point to the remaining staff that they were accomplishing approximately the same amount of work with one seventh of the staff. However, as time went on, the reductions in cost were not what had been originally expected. In addition, when the mainframe and the mainframe staff were gone, so too were the procedures to insure quality. Without the experience of senior staff members, there was no one on hand to recognise the importance of software development controls. Consequently those processes began to deteriorate. End-users were now responsible for their data and didn't know what to do with it. As issues arose, they approached the technical staff directly and requested assistance. Inexperienced programmers, eager to please, developed routines to fix symptoms, and in some cases modified production programs. The lines between testing, production and development were becoming blurred. The software development process was becoming chaotic.

Operational tasks, previously performed by IS and now the responsibility of end-users, were not being performed at all. Tasks such as scheduling when data should be received from customers and recognising when the data was late were falling through the cracks. Immediate steps needed to be taken to improve the quality of internal processes before they began to adversely effect delivery services.

3 An Approach to Quality

Implementation of a quality program is a slow methodical process and this organisation wanted to act quickly before a breakdown in procedures turned into a breakdown of quality. To accomplish this, a plan was developed to re-implement software quality controls and at the same time develop new procedures for those functions that moved out of IS to the end-users. Once these procedures are in place, they will be evaluated and action will be taken to improve them. This evaluation process will be ongoing.

The following outlines the plan for procedure and quality controls.

3.1 Software Quality Controls

Software Configuration Management

Establish controls over the software development process.

- Version Management
- Problem Tracking
- Build & Release Management

Software Baseline

Establish a baseline for the software that is currently in production. This involves identifying all modifications that have been made to production objects, programs, and libraries as a result of the previously described process.

Software Review Board

Establish a software review board to ensure that the functionality of the compiled programs has not been modified during the migration process. Once the baseline has been established, this group will also approve and prioritise any new change requests or newly proposed functionality.

Software Quality Assurance

Establish a formal quality review for all software prior to being placed into production.

3.2 Procedure Quality Controls

This effort was primarily concerned with reviewing operational procedures that had previously been performed by IS, determining if they were still applicable, and if they should remain within IS or if they should be the responsibility of an end-user group.

3.3 Quality Teams

In addition to the quality teams developed in the previous quality initiative, new teams were created to reflect the new technical environment for delivery instructions, made up of members from the technical and user groups.

Define Customer

This process had two groups of customers: Internal users of the software, and external customers that used the service supported by the software.

Define Scope

The newly named Distribution Support Services (DSS) includes all activities necessary to develop the detailed delivery instructions for door to door delivery.

Define Quality Goals

Develop delivery instructions that will reach 98% of the targeted market and to reduce record production costs by 25%.

Define Metrics

Measure delivery data accuracy against postal data.
Measure market saturation by third party surveys.

4 Current Status

The program has been in place now for a few months and is beginning to show results. The Software Configuration Management tools have been purchased, installed, and some training on how to use them has been completed. The process of identifying the most recent versions of production code is progressing faster than anticipated. End-users are grumbling a little about submitting written change and enhancement requests, but everyone involved is readily accepting the software control measures.

5 Conclusion

It is difficult for quality to exist in extremely adverse conditions if the quality process has not been institutionalised. Even then, I am not certain that the outcome of this case would have been any different. The loss of experienced staff contributed greatly to the breakdown of control processes. However, even though their quality initiative had not progressed very far, it had gone far enough to help identify problem areas and provide a structure around which a rebuilding process could begin.

Beside procedural breakdowns, there were management breakdowns as well. Three crucial events took place, that if handled differently, may have averted some of the confusion that followed the migration. There was a point at which the technical staff was unsure of how they fit into the new technical environment. If appropriate measures had been taken to assure the staff that training would be available and that there would be a place in the new environment for them, possibly not so many would have left. When the department head resigned, regardless of the reason, it invalidated the new direction in the eyes of many of the staff. Measures should have been taken to counter that effect. When the experience of the staff reached such low levels, steps should have been taken to ensure that quality controls continued to be used.

In the end, had it not been for SQM techniques, I believe that a few months of chaos could have evolved into something significantly worse.

AN ANALYSIS OF OBSOLESCENCE RISK IN IT SYSTEMS

M. Bradley
Rolls-Royce and Associates Limited
Derby, England

R.J.Dawson
Department of Computer Studies
Loughborough University
Loughborough, England

Abstract

This paper is concerned with technological and functional obsolescence in the context of common computing applications and from a user perspective. Both forms of obsolescence are defined and the effects on software and hardware are discussed, together with the consequence for cost of ownership of IT systems. Supporting data is provided from a number of industrial surveys of commercial off the shelf software.

1. Introduction

There are two principle forms of obsolescence, technological and functional, and these affect the provider and user of products and services in different ways.

Technological obsolescence is a problem that afflicts every delivered product to a degree. The greatest degree of affect is felt in rapidly changing technologies, the least degree in those that change only slowly. Examples of specific products drawn from a slowly changing technology that are affected to only a small degree by obsolescence are bridges and roads. Recent examples of products that are affected to a greater degree by obsolescence are the media used by audio industry. Here, technological advance has inflicted effective obsolescence on various standards of records and tapes over several decades. The effect of technological change and obsolescence is to force new purchase on to the user, whilst providing opportunity to the owners of the incoming technology but disaster to the owners of outgoing technologies.

C. Hawkins et al. (eds.), *Software Quality Management VI*

Functional obsolescence is a secondary problem, generally caused by technological advance elsewhere, and is exemplified by the disuse of sections of Roman roads. Whilst roads in general are not yet obsolete, some roads will become functionally obsolescent as the destination is no longer in use. Functional obsolescence leaves society with a piece of technology that has gone out of use, leaves the owner of the outgoing technology with reduced opportunity, and may provide another technology's owner with a new opportunity.

The crucial difference between the two forms of obsolescence is that the Technological form is caused by a technology owner whilst Functional obsolescence is caused either by the user, or by a different technology entirely - a new technology owner with a step change.

In addition to the typing of obsolescence, there is also an issue on the standpoint from which obsolescence is viewed. There are a great number of articles in the literature written from the position of a manufacturer, [1,2,3,4 & 5], providing advice on the strategies to retain or gain market leadership. There is also advice available from the literature for organisations in the middle of the supply chain [5], where it is clearly important to monitor component obsolescence to avoid both lost sales and obsolete stock..

This paper is concerned with both forms of obsolescence described above in the context of common computing applications, the hardware required and the consequent implications for cost of ownership of IT systems. The context is that of the end user. To provide a common understanding of the subject the following dictionary definitions are provided, taken from Chambers [6]:

Obsolete:' adj. - gone out of use; antiquated; no longer functional or fully developed'

Obsolescent: ' adj. - going out of use; in course of disappearance; tending to become obsolete'

Obsolescence: 'n. - < the process of > ' going out of use'

2. The root causes of obsolescence.

Obsolescence impacts on a PC in a business environment in a number of ways, these are typically caused by:

- software being subject to an upgrade by the software supplier creating technological obsolescence of the earlier software version, and a possible consequential obsolescence of the computing hardware.
- the demands made on a PC are increased within the capability of the software, but beyond the capability of the hardware, creating functional obsolescence of the existing PC.

- the PC is no longer supported because some component is now no longer in production or is no longer in demand by the market place, an example of technological obsolescence.
- the business changes and the software changes in line with the new business, causing functional obsolescence of the software (version), again with a possible consequential obsolescence of the hardware..

An understanding of the likely occurrence of obsolescence is important in understanding the cost of ownership of a PC in the organisation. The first three of these categories are dealt with below, the fourth category is not, as this category is business specific and is best dealt with by carrying out a risk analysis of the business future.

3. The Software Upgrade problem

This form of obsolescence occurs where the software being used is subject to an upgrade by the software supplier and where the upgrade requires an increased specification PC, or where the software is no longer supported at all - withdrawal.

An indication of the volatility of software can be gained from the time between full releases of packaged business software. Clearly, not all releases cause an improved specification to be purchased, but usually the pressure is increased on the business to upgrade. A review of releases for some popular windows based software carried out in May 1997 revealed:

Software vendors generally provide support for software for the current and previous releases only. Some may provide informal support for earlier releases, but may not formally enter into support agreements based on old software versions. This implies that the time between alternate releases is significant. It has also been quite normal for three contiguous PC software releases to move the software from DOS, through Windows™ 3.X to Windows 95™, with significant changes to hardware specification at each change. Based on a total of eight popular software packages with a total of twenty five releases the time between **alternate** software releases is estimated to follow a Lognormal probability distribution, with a Hazard plot as shown in Figure 1. The Hazard function provides an indicator of the change in instantaneous renewal (or obsolescence) rate with respect to time t over the life of a component. The hazard function is normally used to present the 'Bathtub' curve used by many texts to show the normally expected failure characteristics of engineering components.

Figure 1 provides the indication that the instantaneous risk of a second re-release of a single PC software package increases sharply with time, and that the expected time to re release of any one piece of software is 1160 calendar days. This effectively means that if an organisation was fortunate enough to buy a software licence for a business application on the first day that the version was available the chances are that it would be considered obsolete around 1160 days later; or approximately three years. Unfortunately the situation is not normally as clear cut. Most organisations operate a small number of key PC software packages, but also a larger number of lesser used software packages. At the time of writing Rolls-Royce and Associates Limited (RRA) are making widespread use of the Windows™ operating system software and two such packages, WordPerfect and Groupwise . These are the word processing and internal communications packages, accounting for around two thirds of PC use as determined by Green [7]. In total however around twenty commercial off the shelf (COTS) packages are in use, with an approximately random spread of release dates. If two COTS application packages are considered to be running on one PC both of the packages are likely to become obsolete, and if we assume that one of the licences is purchased mid version, the time to obsolescence drops to 870 days. As the number of key software packages increases the expected time to a second re release decreases to around half the expected time for two releases, (580 days or 21 months). What happens in practice is that the dominant packages in an organisation are the ones that decide if the organisation upgrades its hardware. Where specific machines are used with specialist packages eg CAD or statistics, then these smaller number of machines are upgraded in isolation. For the majority of machines the software technological obsolescence causes expenditure on both software and hardware between 580 and 870 days from the joint purchase of hardware and software, despite the fact that the hardware in isolation has a low probability of failure as described by Bradley & Dawson [8]. The minimum cost of ownership solution for a small organisation , given that most businesses are committed to COTS application software, would therefore appear to be the adoption of a policy of software purchase at the release date of the major application(s) used by the organisation, coupled with a hardware upgrade to the most technologically advanced hardware within the organisations budget.

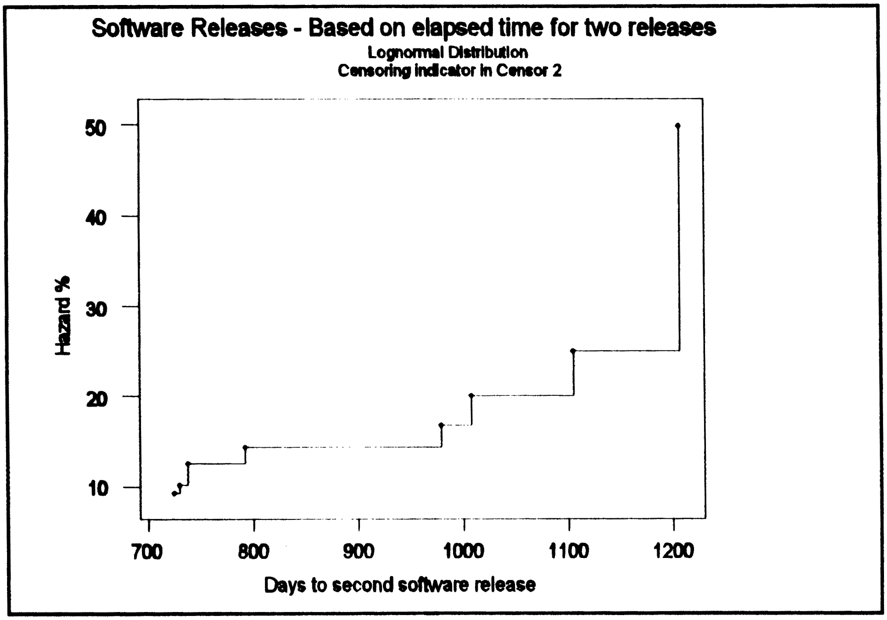


Figure 1 Software Releases - Based on Elapsed Time for Two Releases

This particular form of software obsolescence also has an impact on the software development fraternity; as compilers and their versions change so to must the developer. Developer skill obsolescence is therefore exactly paralleled by the obsolescence characteristics of the software that the developers use. Specifically, this means that there is a likelihood that at two to three year intervals software or software version will change with a corresponding risk to hardware and training.

4. When demand outstrips performance

Obsolescence can occur where the demands made on a PC are increased within the capability of the software but where the PC is not capable of meeting the demand. This can occur where the original PC was not fully capable of handling all aspects of the software; this has happened at RRA even when the PC was the latest specification machine and purchased to 'match' the software, reflecting the software drive for ever better / faster machines. An example of the phenomenon is an initial use of a word processing package to produce simple letters and documents, followed by ever more sophisticated and larger documents as users grow in confidence and awareness of the software .

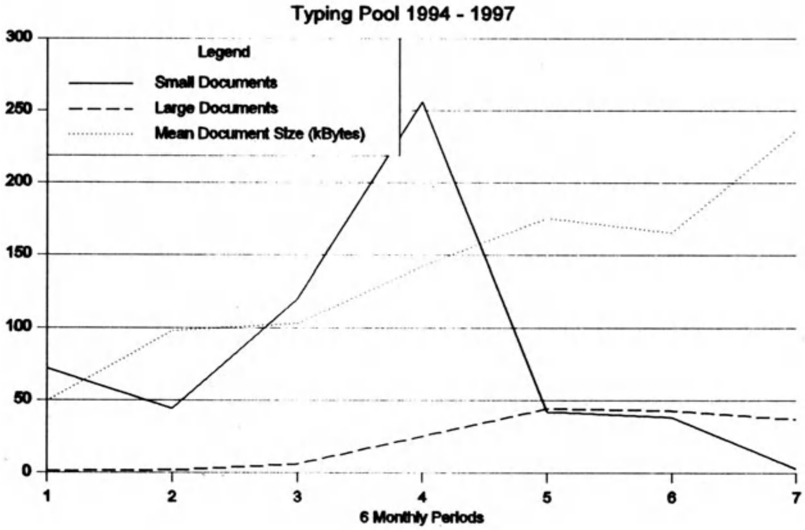


Figure 2 RRA Document Statistics

Data from RRA can demonstrate this. The company implemented a networked WordPerfect word processor system in early 1994. Training was restricted to the typing pool and a proportion of the rest of the Company - around thirty per cent of the total organisation receiving training in the new word processor. The first machines on the network were 80486, 33Mhz machines with 8 Mbytes of RAM. Training was largely to a basic level as it was expected that the corporate helpdesk would handle more complex problems. Figure 2 provides a view of the changing workload experienced by the typing pool in the three and a half years from 1994. That workload was of course dependent on the level of business, although that was relatively constant, with year on year turnover varying by less than ten per cent over the period..

Figure 2 shows that the mean document size being produced by the pool increased steadily over the three and a half years to 250 kbytes. Additionally it can be seen that production of small documents (of 20 kbytes or less) had initially peaked and then almost disappeared entirely after the third year.

This data appears to show the growing confidence of the RRA population of (non-typing pool) WordPerfect users, who, over the period, have gradually undertaken the production of all but the very largest of documents . To move from novice to a very capable and confident user of this comprehensive word processor has taken, we believe, around two years, during which time the PC being used will have been stretched in terms of processor speed and memory. This is demonstrated by Green [7] who recommended in 1997 that ten Pentium 90 machines less than 2 years old should be replaced with large memory Pentium 200's on the grounds of inadequate performance. As Green pointed out RRA is an organisation whose PC use is dominated by the use of WordPerfect, with 50% of PC use restricted to the word processor. The Green recommendation does, however, lead to an organisation with a continuing problem of replacement and where the technology, rather than the organisation appears to be in control.

The impact of this problem is focused on the hardware; with the organisation effectively pressured into the purchase of latest technology that will run the older software more effectively. If the experience with WordPerfect at RRA is typical then the need to upgrade hardware might be expected to occur at around two years after a major word processor version change.

5. A break in the spares supply chain

Effective obsolescence occurs where the PC is no longer supported because some component is no longer in production, or has been overtaken by new technology. In either situation spares to support in-service machines may become unavailable.
The clearest example of this problem, and the one which to date has probably determined PC obsolescence more than any other is the processor installed in the PC. Data from Intel [9] shows that obsolescence of the chip accelerates with each successive design improvement. Figure 3 shows the days between significant design improvements of the 80x86 chip from the original 8086 to the PentiumII. Strictly, Figure 3 shows virtual obsolescence, as the chips can still generally be purchased from the manufacturer. However the supply chain tends to support the current production, and/or the hardware

that is covered by warranty. In the event of a motherboard failure that requires a replacement chip this effectively puts an average limit on the spares availability, and hence machine obsolescence of around three years. Chip improvements, though, have been occurring more frequently recently and this average has fallen to less than two years for the most recent PC sales.

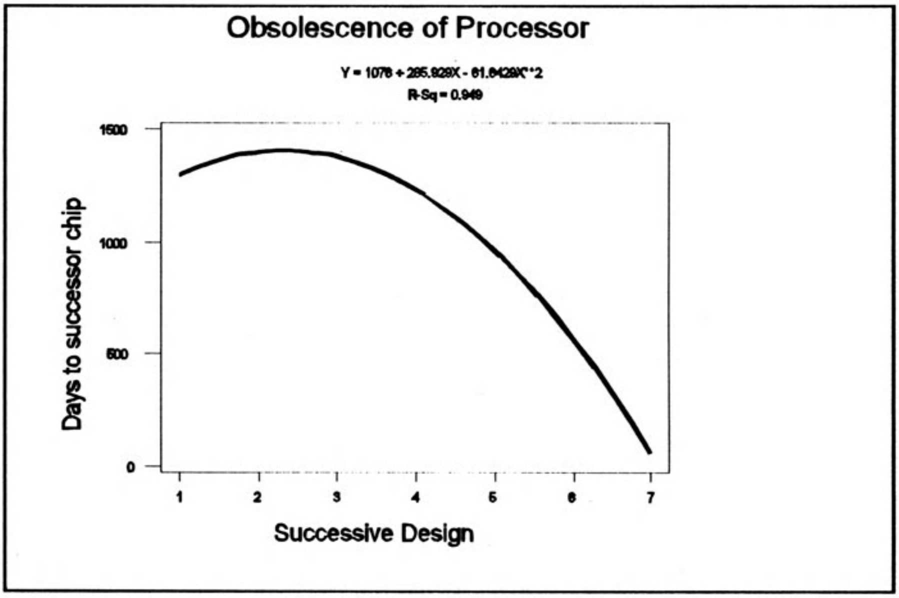


Figure 3 Days between Significant Intel Chip Improvements

6. The problem of the business moving on

The final cause of obsolescence is found where the business moves on, and the software in use has to change This will often be to meet the demands of a (new) prime customer, or because part of the core business has changed. It is an issue that needs careful attention from both the IS / IT management and the main board of an organisation. As it will be different for different organisations it is only possible to generalise on policy for handling the risks as methods for handling these are likely to be the same for all organisations.

7. The need to identify and manage risk

Organisations investing in IT have a need to understand the various risks that can affect the cost of the investment. RRA have had a well developed risk management method since 1994, and largely similar to the principles described by Pressman [10]. This involves identification and assessment of the risks to a project, mapping of the probability and impact of each risk, and putting risk reduction measures in place at the project start.

Risk management is not inexpensive though, and at RRA is therefore usually used only on projects with a value greater than £10M.

The importance of risk management and continuous risk review is illustrated by Sillitoe [11]. In this case the company was requested 'informally' to move from WordPerfect to Word by the main customer. The change if carried out would have involved a move not only to a new word processor, but also to a new operating system, Win 95, which at the time would not run on 70% of the company's PCs. The cost of a change of the full PC population (around 700 were involved) would be in excess of £1M for hardware alone. The risk that might have been anticipated in this case was the obsolescence of the operating software for the extant PC population. Continuous review of the risk would have identified the need for an earlier and gradual migration to a new operating system. At RRA only one example of formal software risk management exists. This was for the implementation of the networked PC / WordPerfect. For this project no account was taken of the impact of obsolescence, as the project was for the implementation of the system, rather than the long term operation of the new network. Risk management needs to be performed more widely for IT systems. Specifically risk management for IT needs to include obsolescence of software and hardware.

8. Conclusions

Obsolescence is a significant root cause of high whole life costs, and yet is not often considered at the planning and implementation stages of software based projects.

Different forms of obsolescence need to be considered and each of these forms an impact on hardware and software.

Both hardware and software can suffer from obsolescence, and this can have an impact on skill obsolescence.

It is possible to gauge the probability and impact of obsolescence of both software and hardware elements of an IT system in the context of both risk assessment and cost of ownership.

9. Reference List

1 Gutgeld.Y., Beter.D.,Are you going out of fashion?, The McKinsey Quarterly 1995 No 3.

2 Nault.B.R., Vandenbosch.M.B.,Eating your own lunch: Protection through preemption, Organisation Science, Vol 7, No 3, May - June 1996.

3 Neff.M.C., Shanklin.W.L.,Creative Destruction as a Market Strategy, Research Technology Management, Vol 40, No 3, May - June 1997.

4 Coredero. R., Managing for Speed to avoid Product Obsolescence: A Survey of Techniques, Journal Of Product Innovation, Dec 1991, Vol 8, No 4.

5 Handfield. R.B., Pannesi. R.T., Managing Component Life Cycles in Dynamic Technological Environments, International Journal of Purchasing and Materials Management. Spring 1994, Vol 30, No 2.

6 Chambers Twentieth Century Dictionary, 1961, W & R Chambers Ltd, Edinburgh.

7 Using Total Cost of Ownership Techniques to Determine Hardware and Software Replacement Strategies at Rolls-Royce and Associates Limited, MSc Dissertation Green.D. September 1997.

8 Reducing the Cost of IT Ownership using feedback from the IT Helpdesk, Bradley,M., Dawson,R.J. in Proceedings of the Software Quality Management Conference Bath, March 1997.

9 Intel web page http://www.intel.com/pressroom/quickref.htm

10 Pressman R.S. , Software Engineering, A Practitioners Approach, McGraw-Hill, London, 1994.

11 Sillitoe J.E., 'WordPerfect versus Word', Internal RRA memo JES/CTO, 23rd January 1997.

Constructing Standards for Cross-Platform Operation

M.A. Heather[1] and B.N. Rossiter[2]

[1] Sutherland Building
University of Northumbria at Newcastle NE1 8ST, UK
[2] Computing Science
Newcastle University NE1 7RU
B.N.Rossiter@newcastle.ac.uk

Abstract. A universal representation is developed, based on the ISO standards for Information Resource Dictionary System (IRDS), with the aim of providing a complete definition of an information system from the physical data values held to the concepts employed for data and function description and real–world abstractions. It is discussed how such a multi–level model can provide quality software for the evolution of information systems by creating an environment where heterogeneous systems can be compared. Current trends towards more structured programming techniques and more disciplined software engineering environments lead to the potential for considerable benefits from using an IRDS approach. This potential, however, will only be realized if a formal underpinning of the IRDS standard is achieved and then only reliably if the *formal* is constructive and the underpinning is enabling not just supporting.

1 INTRODUCTION

Cross-platform operation is not simple, linear or Boolean. It is multi-level, distributed and usually made up of heterogeneous embedded micro-kernels. For consistency, security and safety, highly-reliable local subsystems may be inadequate. For overall quality assurance there must also be reliable cross-platform connections at the global level.

There is both a theoretical and a very practical dimension. If a company's information system fails, the company fails. Liquidation of the whole enterprise often quickly follows. Robustness is therefore critical particularly in high-consequent computing. The cross-platform software must be able to cope with demands that may not be specifiable in advance. The functionality may be supporting or it may be enabling. If it is supporting it is self-organizing and naturally reacts to the unforeseen. If merely enabling it can only respond as pre-specified. For reliance across levels in the real-world the linking software needs to provide either a natural closure or full information of all limiting constraints.

C. Hawkins et al. (eds.), *Software Quality Management VI*

2 RATIONALE

There is a problem in dealing with layers or levels and transitions between levels in developing reference models. Cross-platform operation needs reference models rather than just standards. This does not mean that standards are superseded but rather subsumed. For example the ISO family of OSI standards[12] are widely accepted and successfully used as a convention for cooperative work but their value is limited to the syntactical level. For while there is internal consistency in the standard there is no guarantee that the application of the standard will result in a self-consistent system. This need not cause too much concern for implementers in a local system where everything is under their own control. A programming language or a LAN protocol is a close enough approximation to a closed system where little difficulty may be experienced in practice. However, as soon as any kind of openness or independent autonomicity is introduced, another level appears requiring closure at an even higher level. In terms of logic, higher-order is needed: first-order logic will just not do to connect between levels. The result is a need for a meta-meta level. Otherwise a cross-platform system is inadequate.

What is needed is a reference level in its most abstract form which can give this provable ultimate closure. Mathematics gives us this third-level closure through constructive methods for defining the reference model for applications like client/server systems. This need has been recognized to a limited extent by standards bodies who have produced reference models which relate local standards across a number of levels. However, true reference models are still few and far between.

It is important to bear in mind what is meant by a reference model[2, 16]. If we consider the client/server context as an example, the reference model for OSI is not itself the set of protocols for a communications system. It is a framework for the identification and design of protocols for existing or for future communications systems. It enables engineers to identify and relate together different areas of standardization. OSI does not imply any particular technology or methods of implementing systems. Put simply, the reference model helps engineers to design protocols for computing communications systems.

This multi-level context of the reference model also raises problems in the development of the theory which is always needed to give underlying confidence in matters of consistency and reliability, etc. Formal methods where limited to set-based approaches with axiomatic sets and first-order logic may well be satisfactory in theory for localized systems. However, in practice first-order predicate logic involves considerable mental effort, is difficult to learn and is viewed as an academic language whose application is to some extent an art[6, 8]. Relational and functional theories are much more suited to representation of real-world activities and relationships[13] but have been rather neglected.

In this paper we seek to show that the IRDS has a constructive formal basis and can be relied on for this reason. It is therefore a valid method for formalizing a cross-platform perspective of the construction and implementation of information systems.

3 LEVELS in the UNIVERSAL REFERENCE MODEL

As indicated above, it is always necessary to go one level up to address the physical level from a logical standpoint. A minimum requirement is to know names, constraints, typing, etc. In modern client/server systems for example, this is the function of the system catalogue which for a relational database system, for example, includes descriptions of the relation names, attribute names, attribute domains or types, key i.e. identifying attributes, other constraints, views, storage structures and indexes. In a network database, information held in the catalogue includes descriptions of record types, set types and physical organization. In a COBOL program, the data division is held in a catalogue giving information on record types, field types and their pictures. In C++ or Java, classes define data structures, methods and their abstractions.

For client/server systems, the specification includes:

- the overall layout of the system including mainframe-centric or network-centric organization;
- hardware components;
- the segmentation and allocation of functions between client and server;
- distributed database design;
- participating client/server operating systems (Unix, Windows 95, Windows 5.x, Windows NT, XWindows, AIX, OS/390, OS/2, etc);
- selection of middleware (CORBA, ODBC, COSE, GUI builders, FTAM, CDE, etc);
- declaration of network technology (SNA, ISDN, LAN, internet, ETN, TCP/IP. EDIFACT, etc);
- client and server program structure (e.g. documentation of Scripts).

For instance nearly all system catalogues today are *active* in the sense that they are a dynamic automatic source of naming and typing information for programs accessing the system (rather than a passive static reference). This makes the catalogue the cornerstone of the information system, often being subjected to a very high rate of searching activity. Clearly a major advantage of active catalogues is that system type changes are made once in the catalogue rather than replicated through many source programs.

However, a logical level is incomplete without an associated formal reference level to describe how it represents real-world entities, properties, relationships, typing, etc. This reference level deals with meta-meta data. Another way of looking at this level is to view it as dealing with the policy of the model: why is a particular construction in the model available and what semantic capability does it handle?

This comparison of logic and principles is performed against a reference or neutral layer which contains a superset of the facilities for a particular platform or application model. The reference layer contains as data the principles for data and activity representation and for each application the concepts which capture these principles. There is therefore a mapping from principles to schema intension for each application model in turn.

In relational systems with all data held in tables, the meta-meta level is implicitly part of the model. The freeness of the object-oriented paradigm, on the other hand, means that the meta-meta level needs to be constructed explicitly to control the representation of classes, objects, properties, inheritance, composition, methods, etc. In a COBOL system, *wrapper* constructions are increasingly used to provide encapsulation of programs with pre-determined interfaces to emulate object constructions. Other examples are the class of shell scripts and functions like Korn shells and similar patches for unix or motif widgets in XWindows. However, these shells and wrappers are at the meta level: the meta-meta level would describe the purpose of wrappers and the facilities used for their construction. This also seems to be the crux of the problem with Java mentioned above. Applets are at the meta level and it is the metameta level which gives the overall reliability.

These are particular examples of the message level which provides a top layer of awareness which has to be developed further to deal with the widest possible type of information systems. It was for this kind of purpose that the ANSI Standard Reference Model (X3.138) was developed in the 1980s[4], emerging in 1993[10] as an associated part of the standard for a framework for Information Resource Dictionary System (IRDS) developed in 1990 and 1993[9]. Among early language bindings in IRDS were specifications for Pascal, COBOL, C and SQL. Language bindings can in principle be defined for any language. However little use seems to have been made of the IRDS but this may be because only recently has there been much need to cross levels in systems work.

In the meantime attention has been concentrated on the OSI seven-layer model for open systems because it provides a potential means for commercial suppliers to provide compatible components for different systems. However, while these may be compatible, there is no guarantee that they are consistent. This is because the OSI set of standards together provide a reference model but not a universal reference model. The IRDs on the other hand has the internal characteristics of a universal reference model.

The four levels of the IRDS can be shown to capture the universal nature of the reference model because it can be demonstrated that these can be constructed formally. Before we attempt to show this we should examine closer the four levels and their inter-relations to see how they bear out the discussion so far.

4 The INFORMATION RESOURCE DICTIONARY SYSTEM IRDS

Before embarking on a full formal description of the IRDS, some understanding and informal insight into its interpretation would be useful. The IRDS is constructed on four levels. Each level taken with its adjacent level acts as a *level pair* so that there are three level pairs across the four levels. This means that each point at each level is directly related to a point at the other level in the *level pair*.

The top level is the Information Resource Dictionary Definition Schema (IRDDS), in which concepts relating to policy and philosophy are defined. For example, object-oriented abstractions are to be declared at this level. In principle, only one instance of an IRDDS need be defined for a platform. In a coherent system there can be only one collection of such concepts. With the open-ended nature of object-oriented structures, however some extensibility may be required.

level	instances	interpretation	example (relational model)
1. meta-meta /policy level	usual abstractions (aggregation, composition, inheritance, association, etc)	mission (natural concepts)	real-world abstractions
2. meta/ operational level	abstractions at policy level	organizational (high-level operational/analytical tools)	available constructs
3. intension level	network features and functions	formal declaration, schema and labels	data names
4. extension level	data values satisfying the intension	the information itself	data values

Fig. 1. Interpretation of Levels in the IRDS

The second level is the Information Resource Dictionary Definition (IRDD) in which schema facilities are defined. The context of each platform will have its own IRDD definition. For example a COBOL IRDD would declare that record-types were an aggregation of single- or multi-valued data field-types while one for SQL would declare that table-types were an aggregation of single-valued data fields.

The third level is the Information Resource Dictionary (IRD) which defines the intension for an application, giving names and constraints. There will clearly be many intensions defined in an organization, one for each application. Names, types and other constraints will be given to data objects, network connections, protocol names and signatures, server and client functions, etc.

The fourth level is the Information Resource Data (APP) which gives the extension, the data values. There will be one extension for each intension, the values being consistent with the names and constraints of the intension. Data values may be simple objects as in SQL or complex objects as in computer-aided design and multimedia systems.

level	IRDS standard	top-down mapping	example of level-pair
1	concepts		aggregation
	↓	Policy	↓
2	constructs		table
	↓	Org	↓
3	intension		name of table
	↓	Data	↓
4	extension		data value

Fig. 2. Mappings between Levels in the IRDS

One instance of the Information Resource Dictionary System represents one platform, paradigm or model. Take as an example the relational model. Level 1 would be real-world abstractions; level 2 would be the constructs available; level 3 would be the data names; and level 4 would be the data values. The four levels with their terms, instances, interpretation and the corresponding components of the relational model can be summarized in the table of Figure 2.

Between each level the mapping, between the *level pair* enables data at one level to be related to data at a lower level. For the same example of the relational data model, the level pair between levels 1 and 2 would be the association of an abstraction (say aggregation), with a constructive facility in the model (say table in this case); the level pair between 2 and 3 would be the association of a construction in the model (say table, with the name of a table) and the level pair between 3 and 4 would be the association of a data name with a data value. Figure 2 shows schematically the mappings between the various levels.

Between each level the mappings are strictly defined by their starting and terminating points in the respective levels. These may not be immediately obvious in the original standard but they are brought out in the informal diagram of Figure 3 together with more explicit interpretations of the levels. In particular it should be noticed that the interpretations of the mappings can only be appreciated by considering both directions for each respective mapping. The bottom-up mappings are described in the formal model. The top-down mappings in Figure 3 are as follows:

- Between levels 1 and 2 (IRDDS and IRDD), there is the mapping *Policy* acting as a level pair. This level pair exists only in IRDS-type systems in which constructive facilities in a system are related to real-world abstractions. For example, *Policy* would indicate how a network-centric capability is made available in a particular approach.

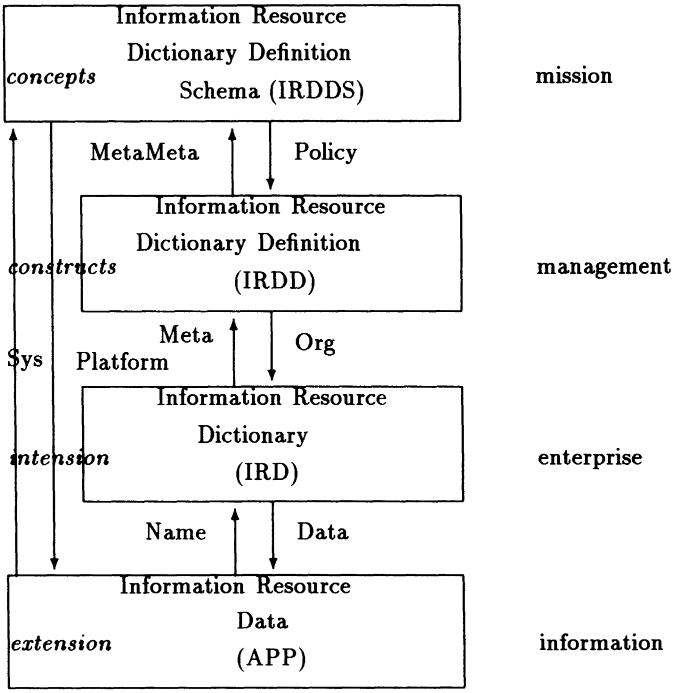


Fig. 3. Interpretation of IRDS in Schematic Form

- Between levels 2 and 3 (IRDD and IRD), there is the mapping *Org* acting as a level pair. This level pair provides a standard data dictionary function of, for instance, saying which tables are available in a relational system or which servers are available on a network.
- Between levels 3 and 4 (IRD and APP), there is the mapping *Data* acting as a level pair. This level pair can be thought of as the state of the art of an information system: to link values to names so that data can be addressed by name rather than by physical location.

- Between levels 1 and 4 (IRDDS and APP), there is the mapping *Platform* acting as a level pair. This level pair short-circuits the navigation through four levels by giving a direct mapping from real-world abstractions to data values. The use of this mapping is described later.

The IRDS standard is the basis for relating heterogeneous systems across platforms, that is systems based on different paradigms. While there is only one instance of the top level (the IRDDS), this level is extensible and new concepts and abstractions can be added as desired. From the point of view of client/servers, the IRDS provides the ability to run an organization with many different paradigms all integrated through the type of structure shown in Figure 3. The critical mapping is *Platform*, that is the arrow from IRDDS to APP, relating concepts to values. By determining this mapping for all types of system, the problems arising in re-engineering are avoided to some extent as all types of approach to information systems can be accomodated and run in an integrated fashion. The way that federated database systems are managed[3]is an example of this approach.

The next task is to formalize the diagram in Figure 3 so that a sound scientific basis can be developed for the IRDS model to handle heterogeneous systems.

5 FORMALIZING the IRDS

Constructive mathematics attempts to develop logically what works in practice and can provide the necessary universality for interoperability of heterogeneous data systems with consistency and quality assurance in the real-world. Category theory[1, 14] is particularly appropriate for modelling multi-level relationships for it is essentially concerned with links between objects. It has been shown, for instance, to cover adequately dynamic aspects in hypermedia[7].

Category theory provides a universal construction for formalizing information systems. It is this uniqueness that provides the universalness that provides the basis of a general consistent system. An example is now given for a prototype information system focusing on the aspect of a cross-platform system as a heterogeneous distributed database relying on the categorical product construct as a data model. In this approach, each class definition can be identified as a collection of arrows (functions) forming a category **IRD** and each family of object values conforming to a particular class definition as a category **APP**. The mapping from the intension (class definition) to extension (object values) is made by a functor *Data* which enforces the various constraints specified in **IRD**. Category **IRD** is the intension corresponding to the third level in IRDS and **APP** is the extension corresponding to the fourth level.

In reality the intension category **IRD** is a family of categories, representing definitions of classes, associations (relationships) and coproduct structures indicating inheritance hierarchies. The arrows within it may be methods as in object-based systems, network connections between clients and servers, logical connections as in network databases, or functional dependencies as in relational database schemas. It should be emphasised that categorical approaches naturally include procedures and functions through the underlying arrow concept ensuring that both structure and activity can be modelled in a multi-level manner. The category **APP** is also a family of categories, representing object values and association instances. The functor *Data* mapping from the intension to the extension not only connects a name to its corresponding set of values but also ensures that constraints specified in the schema, such as functionalities of relationships, membership classes and functional dependencies, all hold in the extension.

$$\begin{tikzcd}[column sep=huge, row sep=huge]
\mathbf{IRDDS} \arrow[r, shift right, "Policy"'] \arrow[d, shift left, "Platform"] & \mathbf{IRDD} \arrow[l, shift right, "MetaMeta"'] \arrow[d, shift right, "Org"'] \\
\mathbf{APP} \arrow[u, shift left, "Sys"] \arrow[r, shift right, "Name"'] & \mathbf{IRD} \arrow[u, shift right, "Meta"'] \arrow[l, shift right, "Data"']
\end{tikzcd}$$

Fig. 4. IRDS Levels in Functorial Terms

It is relatively straight-forward in category theory to extend the intension and extension two-level structures in a universal manner to handle the four levels of IRDS. In categorial terms each of the four levels of IRDS is defined as a category. Between each level there is a higher-order function, a functor, which ensures that certain consistency requirements are met in the mapping between the source and target categories. The abstractions level (top) is a category **IRDDS** which defines the various abstractions available for modelling real-world data. The next level is a category **IRDD** defining the various construction facilities available for representing abstractions and data in a particular system. There is therefore, for one instance of **IRDDS**, many instances of **IRDD**, one for each type of model (relational, network, COBOL, etc).

The data functor (level pair) *Policy* maps target objects and arrows in the category **IRDDS** to image objects in the category **IRDD** for each type of system. This mapping provides at the meta-meta level the data for each kind of system, that is to say how each abstraction is to be represented. We also label the functor pair *Org* relating for each system the constructions in **IRDD** with the names in a particular application in **IRD**. Combining these new constructions with the product ones above gives the following direct and universal representation of IRDS as shown in Figure 4

In category theory, the above is a composition of functors with *Platform* as the overall functor from **IRDDS** $\longrightarrow$ **APP**, such that for each type of information system the following composition holds:

$$Platform = Data \circ Org \circ Policy$$

as does its dual:

$$Sys = MetaMeta \circ Meta \circ Name$$

In order to relate concepts across platforms, we need to compare the functors *Platform* : **IRDDS** $\longrightarrow$ **APP** for each of our COBOL, network, relational, object-oriented systems, etc. This comparison is a natural transformation.

6 IMPLEMENTATION

For experimental testing purposes, an example prototype information system was developed called the Categorical Product Data Model (CPDM) which can formalize, for instance, the object-relational model[15]. The purpose of this prototype was to test the theoretical representation given in section 5 and to examine the ease with which such a representation could be implemented on the computer.

A prototype[11] of CPDM was realized using the platform of P/FDM[5], an implementation of the functional database model developed at Aberdeen University. Categories and functors were coded so that the two categories **IRD** and **APP** could be defined together with the mapping *Data* between them. In the actual work described in [11] the categories **IRD** and **APP** are called **INT** (for intension) and **EXT** (for extension) respectively. The universal nature of category theory means that it is a simple matter to extend the product data model CPDM, defined as a two-level structure, to handle the four levels of IRDS. Further work is planned to explore further the details of the management of the four-level architecture for assisting in the control of cross-platform operations.

7 DISCUSSION

A formal universal description, based on the ISO standards for Information Resource Dictionary System (IRDS), can therefore be developed and implemented to provide a complete definition of an information system from the physical data values held to the concepts employed for data and function representation and real–world abstractions. Such a multi–level formalism model can be used to control the evolution of information systems by creating an environment where heterogeneous systems can be compared for cross-platform performance. Current trends towards more structured programming techniques and more disciplined software engineering environments can very usefully exploit the IRDS approach. This potential which the IRDS possesses, however, will only be realized for consistent cross-platform operation if a formal underpinning of the standard is achieved.

The constructions of the IRDS in systems offer a certain path for improving our ability to deal with heterogeneity. The universal basis provides a predictable and provable behaviour which is such an important aspect if heterogeneous information systems are to be reliable and productive. Essentially the formal basis for IRDS assists in developing a reference model to relate policy statements in different models and environments with consistency and integrity. This is the crux of interoperability.

A categorial IRDS achieves greater power than that envisaged in the original standards. For instance Spurr[17] comments on the difficulties of using the IRDS standard dictionary with CASE tools because of the lack in IRDS of a natural way of modelling object structures. Our work shows the natural correspondence between categorical databases and object structures making possible a complete and faithful representation of the object-oriented paradigm across the four levels[11].

A possible difficulty is that this approach may require a complete knowledge of all features of every type of system available. The top level IRDDS is always available but an IRDD may not be available for all paradigms, particularly if usage of the paradigm tends to be idiosyncratic. Even with a clearly-defined IRDD paradigm, there may still be difficulties if, in the original system, there are problems with incomplete or conflicting schema information or code has been written bypassing the conceptual tools of a system. Nevertheless there is no difficulty that cannot be solved (at least in principle) by going to higher levels because of the infinite closure property of the four level approach.

We therefore see these techniques being more useful in coping with well-defined systems, even if they are using outdated methods, than poorly-engineered historical software where the case for a complete rewrite is obviously stronger. The improvements in languages, for instance Microfocus COBOL and Visual BASIC, and the greater emphasis on formal schema design in current software engineering both indicate that future problems will be more amenable to the four-level approach.

It is for this reason suggested that current trends towards more structured programming techniques and more disciplined software engineering environments should be leading to greater use of this four-level IRDS standard with its formal underpinning. Our conclusion is that theory shows that for situations where there are heterogeneous systems as in cross-platform operation, client/servers, legacy software, etc which have to be coherently designed and implemented, there is no other way forward than by this four level closure path or with some equivalent framework as this categorical approach. For such systems presuppose universal techniques to ensure consistency and integrity across time and distribution.

References

1. Barr, M, & Wells, C, *Category Theory for Computing Science*, Prentice-Hall (1990).
2. Clements, Alan, *Standardization for Information Technology,* BSI Catalogue no: PP 7315, at page 26 (1987).
3. Fiddian, N J, Gray, W A, Ramfos, A, & Cooke, A, Database Meta-Translation Technology: Integration, Status and Application, *Database Technology* 4, 259-263 (1992).
4. Gradwell, D, Developments in Data Dictionary Standard, *Computer Bulletin*, September 1987.
5. Gray, P M D, Kulkarni, K G, & Paton, N W, *Object-Oriented Databases: A Semantic Data Model Approach*, Prentice Hall (1992).
6. Gries, D, The Need for Education in Useful Formal Logic, *IEEE Computer* **29**(4) 29-30 (1996).
7. Heather, M A, & Rossiter, B N, Content Self-awareness in Distributed Multimedia Publishing: the Need for a Unifying Theory, in: Third International Workshop on Principles of Document Processing (PODP'96), ed. Nicholas, C, & Wood, D, *Lecture Notes in Computer Science* **1293** Springer-Verlag 59-86 (1997).
8. Holloway, C M, & Butler, R W, Impediments to Industrial Use of Formal Methods, *IEEE Computer* **29**(4) 25-26 (1996).
9. Information technology - *Information Resource Dictionary System (IRDS) framework*, Standard ISO/IEC 10027 (1990); 10728 (1993).
10. Information technology - *Reference Model of Data Management*, Standard ISO/IEC 10032 (1993).
11. Nelson, D A, & Rossiter, B N, Prototyping a Categorical Database in P/FDM. *Proceedings of the Second International Workshop on Advances in Databases and Information Systems (ADBIS'95)*, Moscow, 27-30 June 1995, Springer-Verlag Workshops in Computing, edd. J. Eder and L.A. Kalinichenko, ISBN 3-540-76014-8, 432-456 (1996).
12. Standards dealing with OSI (Open Systems Interconnection) include BS ISO/IEC TR 9571 to 9596 and BS ISO/IEC TR 10162 to 10183.
13. Parnas, D L, Mathematical Methods: What we Need and Don't Need, *IEEE Computer* **29**(4) 28-29 (1996).
14. Poigné, Axel, Basic Category Theory, in: Abramsky, S, Gabbay, Dov M, & Maibaum, T S E, (edd), Handbook of Logic and Computer Science, **I** Background: Mathematical Structures, Clarendon Press, Oxford pp.416-640 (1992).
15. Rossiter, B N, Nelson, D A, & Heather, M A, *The Categorical Product Data Model as a Formalism for Object-Relational Databases,* Technical Report, Computing Science, Newcastle University, no.505, 41pp (1995).
16. Rossiter, B N, & Heather, M A, *Data Modelling for Migrating Information Systems*, chapter 1, in: Legacy to Client/Server – Have You Chosen Wisely?, ed. Booth, A, Unicom, London 1–12 (1996).
17. Spurr, K, CASE Tools, Does the ISO Standard IRDS provide Sufficient Support? in: *Fourth Generation Systems*, ed. S.Holloway, Chapman and Hall, Unicom 36-47 (1990).

This article was processed using the LaTeX macro package with LLNCS style

The Practical Use and Abuse of Software Metrics

Author
Kawal Banga

Abstract:

"You can't control what you can't measure" is a particularly pertinent comment when directed at software development organisations. Most software organisations have at some time or other implemented a metrics programme. Some of these organisations have managed to make significant improvements in software development. However, experience shows that most metrics programmes either fail completely or at most achieve limited success. One of the major contributing factors for this lack of success is that managers, particularly senior managers and directors, do not really understand how to gather, interpret and use metrics.

1.0 Introduction

In 1990 Rubin [1] reported that out of 300 major US IT companies that had implemented measurement programmes, only sixty were successful. In 1995, Rubin [2] conducted research that revealed that most IT measurement programmes fail within 8-15 months of initiation. Every metrics guru will have his or her own list of reasons why so many metrics programmes fail i.e. a list of do's and don'ts. The author's experiences lead him to believe that one of the major contributing factors for this lack of success is that senior managers and directors, do not really understand how to gather, interpret and use metrics. This paper will give real live examples of the use and abuse of software metrics.

2.0 Quality of Data

There is a school of thought that states that if there exists a metrics database, then it must be useful. However, this is far from the truth.

2.1 Omissions from Effort Data

Jones [3] lists the following as the commonest omissions from historical data, along with the corresponding magnitude of the error:

- Unpaid overtime by exempt staff (up to 25% of reported effort)

C. Hawkins et al. (eds.), *Software Quality Management VI*

- Charging time to the wrong project (up to 20% reported effort)
- User effort on projects (up to 20% reported effort)
- Management effort on projects (up to 15% reported effort)
- Specialist effort on projects (up to 15% reported effort)
- Effort spent prior to "turning on" the project tracking system for the project (up to 10% reported effort)
- Inclusion of non-project tasks (up to 5% reported effort)

This gives an overall error magnitude of 110%.

The author is aware of one organisation within which it is routine for projects that have exceeded, or will soon exceed, their authorised budgets, to record effort against projects that are on track to under-spend their budgets. One particular project had already had 3 false starts and had spent £1.5 M even before there was a fourth attempt.

Another common theme appears to be the deliberate omission of low productivity projects and/or projects that are cancelled, despite having utilised a large proportion of the IT departments overall resources.

2.2 The Elusive Staff-month

The author interviewed 19 project managers within the same organisation and received the following definitions of a staff month: 20, 18, 18.5 and 17 days. Furthermore, the definition of a staff day was 8, 7, 7.5, 7.4, 5 or 4.75 hours. In this same organisation the author discovered that some staff were recording gross hours, and others net hours on the effort tracking system.

In another organisation, the senior managers tried to compare the productivity of two projects, without realising that one team consisted wholly of contractors who worked 8 hour days and the other team consisted mainly of permanent staff who worked 7 hour days. This particular exercise was aimed at demonstrating that contractors were more productive than permanent staff and permanent staff should be replaced. Not only was the exercise flawed, it was very divisive and resulted in a lot of staff dissatisfaction and uncertainty.

2.3 Misrepresentation of Data

The author has come across several incidences of organisations deliberately misrepresenting data. This of course further erodes the quality of such data.

In one organisation managers deliberately excluded overtime effort (an average of 15%) so that the productivity figures were higher. Furthermore, staff were willing to work the extra hours unpaid for fear of becoming the next batch of individuals targeted for down-sizing within the organisation. This meant that future estimates based on this data were automatically flawed i.e. under-estimates of 15%.

This same organisation decided not to move from version 3.2 to 3.4 of the IFPUG counting rules as this would substantially lower their productivity rates. They stopped quoting the IFPUG version number whenever quoting the figures in the public domain. Yet another organisation applied the IFPUG supplied 3.4 to 4.0 conversion formula to their entire portfolio of applications, despite the fact that their were many applications that were purely batch, which would have the same FPs whether counted under the 3.4 rules or the 4.0 rules.

2.4 The Accuracy and Use of FP Counts

Research, and experience, show that FP counts are only accurate to within plus or minus 10% [4]. Therefore an organisation developing 10,000 FPs in a year, at say a cost of £500 per FP, could be up to £0.5M out. Thus budgets, outsourcing costs/deals and estimates would be severely impacted.

Another subtle difference is between FPs developed and FPs delivered i.e. the level of re-use. For example, an off the shelf package may be 10,000 FPs but only cost ten staff months of effort to customise and install. This gives a productivity of 1,000 FPs per staff month (FPs/SM), which of course is ludicrous. A more realistic example might be where a project re-uses the functionality of an existing application. Similarly, new development, enhancement and bug fix (maintenance) projects all have different profiles.

A particular organisation appeared to show an upwards trend over many years with regards to productivity. However, it suffered from many of the problems listed above. Also, in the latter years, a lot of package implementations and small projects have helped to artificially boost the productivity rates.

2.5 Flawed Defect Trends

Many, if not most, organisations tend to show defect trends without too much thought. Many of these use such charts for political reasons, to demonstrate to internal management or perhaps external customers how wonderful they are at developing quality software. However, what the customer may be interested in is the severity 1 (system unusable) and severity 2 (some functions unusable) defects rather than a consolidated defects chart. The author has seen such a trend chart in two organisations where the actual trends in severity 1 and 2 defects were getting worse, but the overall downward trend was due to severity 3 and 4 defects getting less frequent. Furthermore, there is no attempt to normalise the data i.e. defects per FP. It may be that they are getting smaller numbers of defects simply because they are writing less or smaller volumes of code.

A similar naive use of such data is found when a metrics programme reports the number of requirements changes. Typically, the aim is to reduce the number of requirements changes during the project life-cycle. Again, this can be potentially meaningless, if not misleading, unless normalised data is used. For example, which is better, a project with 10 requirements changes or one with 20? How about a 1,000 FP project with 10 changes and a 1,000 FP project with 20 changes? Even this is not sufficient information. What is important is not the numbers of change requests, but requested changes measured as a proportion of the original project size. For example a 1,000 FP project with 10 additional requirements totalling 150 FPs gives 15% requirements creep. Another 1,000 FP project with 20 changes that total 100 FPs gives only 10% requirements creep. This is fine for setting targets for project requirements creep e.g. 10% or less. However, if we are interested in determining the total costs of requirements creep for a particular project, then the absolute number of FPs attributable to the scope creep is the required measurement. Thus the specific context in which the data is to be used will determine the way the data is presented and used.

2.6 Beware of Benchmark Data

The typical cost of an external benchmarking exercise through organisations such as CSC Index or Gartner Group is around £30K. However, organisations that make use of such benchmarks should be aware of all of the issues discussed regarding the quality of data. The author is aware of one organisation that put forward its best projects. It was of course no surprise that they came top in their industry sector. However, this backfired on the IT organisation. A director of one of the business units that was supplied by the IT organisation queried why they claimed average productivity's of 25 FPs/SM, but were using a value of 5 FPs/SM when estimating a new project.

Another common scenario is as follows. A benchmark shows an IT department in good light. They introduce new tools, techniques or methods and discover that productivity has fallen dramatically. This is of course as one would expect, as the introduction of new methods of working requires training and familiarization before staff fully realise the benefits. The IT department either can't explain the figures to the board/senior management, or think that they won't be keen on learn that the £500,000 spent on new tools has not delivered the promised 25% productivity increase. So, they try to bury the results of the latest benchmark.

One organisation was told repeatedly by its benchmarking company what it must do to further improve its performance. The same messages came out at the end of each subsequent benchmark exercise. However, the senior management team did not follow through on these actions. They were only concerned with using the data to show the main board how wonderful they were at delivering IT solutions.

When the benchmarking company does not have sufficient data for the specific industry sector, they simply use data from across different industry sectors. This again is flawed as different industries tend to have different productivity and quality profiles.

Another problem area is the source of the FP data itself. Not all companies use FPs, so they either estimate the FP counts or derive them from lines of code using backfiring. Backfiring is only accurate to plus or minus 25%, and in some extreme cases can vary by more than 50% [3]. Some organisations only sample an application and scale the count up accordingly. Such approaches are prone to high error margins. Even if the source of the data is FPs, which version? IFPUG 3.2, 3.4 or 4.0 or is it Mark II? In one organisation both IFPUG and Mark II versions are used, depending upon who does the count. Unfortunately the significance of the difference is lost on many of the managers and project managers who do not have the necessary metrics backgrounds.

Even though Lines of Code (LOC) has been discredited as an appropriate measure of size, many organisations, academics and authors insist on using it. For example, Ashley [5] quotes that the "Space shuttle achieved 0.1 defects per KLOC during first 3 years of operational use". What does this mean? How can this be used as a comparative measure without knowing the particular language(s) used? A system with 0.1 defects per KLOC of assembler would generate at least a magnitude more errors than a system with 0.1 defects per KLOC of a 4GL, both systems delivering the same amount (measured in FPs) of functionality.

The typical metrics database from a medium to large size organisation will consist of a variety of projects - small to large in size, varying productivity rates, data from many different years, a mixture of new developments and enhancements projects and a variety of different languages - 2GL, 3GL, and 4GL with some a mixture of these. Without this detailed understanding, the data is almost useless. Unfortunately, many organisations use averages derived from such databases and thus are not comparing apples with apples. For example, Jones [6] demonstrates clearly the need to distinguish between new development, enhancement and maintenance projects in figure 1. The productivity profiles are quite different.

Another common problem is the omission of effort relating to projects that get cancelled. An organisation may have a high productivity rate for those projects it does deliver, but may have a high incidence of cancelled projects. In order to make a true comparison between organisations, the organisational productivity i.e. Total FPs delivered divided by total effort of all projects should be calculated.

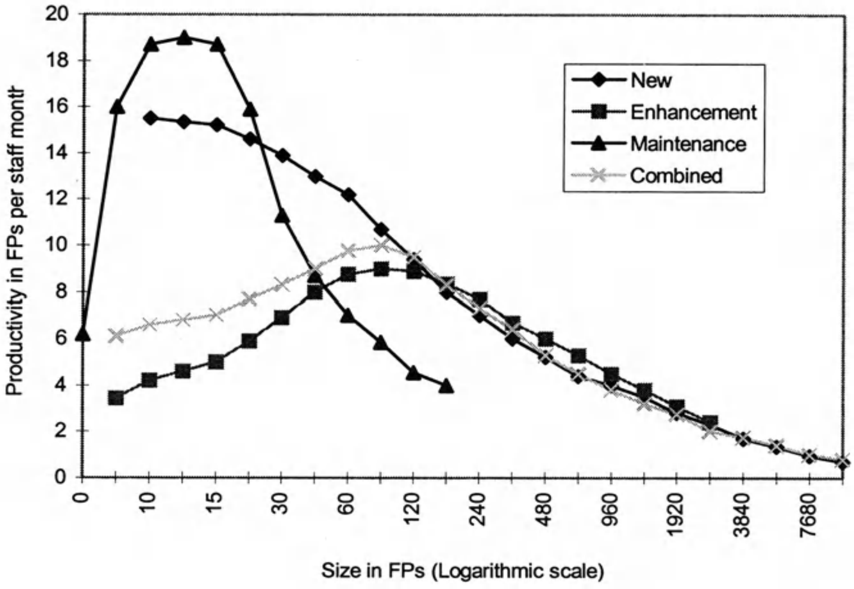


Figure 1: Productivity Profiles for New, Enhancement and Maintenance development (Source Jones [6], reproduced with permission)

3.0 Practical Uses of Metrics

Now that some of the problems associated with metrics have been discussed, it is time to look at some of the practical uses of metrics and particular approaches in using metrics.

3.1 Documentation

Boehm and Jones [7] have both noted that for software development over half the effort expended is in producing documentation. However, few if any organisations have data on the amount of documentation produced by their software projects, and therefore how much time needs to be planned for reviewing the documentation produced during the software life-cycle. The author has gathered some metrics shown below which have been used for this purpose.

Documents have been classified into Low (L), Medium (M) or High (H) complexity as described in figure 2.

Type	Examples
L	Memos, status reports, checklists, simple process documents, application user guides
M	Project/Programme Plans, Quality Plans, Project/Programme Definition Reports, End of Phase reports, Post Implementation Review reports
H	Technical reports, technical standards, technical tool user guides eg CASE tools and documents that include: complicated tables, diagrams and formulae, complicated statistical analysis, cross-references to other documents and multiple appendices

Figure 2: Documentation Classification

Figure 3 shows the range of times to review L, M and H complexity documents in minutes per A4 page (excluding front page, revision information and table of contents). The range is shown to indicate the potential variation. However, only the average values should be used for estimating purposes.

Range	**L**	**M**	**H**
Minimum	0.25	0.75	3
Average	**0.5**	**1**	**5**
Maximum	0.75	3	15

Figure 3: Range of times taken to review documents in minutes per A4 page

3.2 The Business Scorecard

The Business Scorecard is a good tool for implementing a metrics programme. A typical approach is described below. All scorecards have many indices, such as shown in figure 4. This ensures that a consolidated view of performance is considered, as opposed to attempting to assess performance from just one measure. For example, it is quite possible to improve productivity dramatically, but at the cost of lower quality and lower customer satisfaction. This scenario is avoided by the use of the scorecard approach.

Index	**Description**
Financial Index	"Managing finances, unit costs and cost flexibility."
Quality Index	"Delivering products and services to agreed specification."
Productivity Index	"Improving value for money and capability."
People Index	"Managing the skill base and staff satisfaction."
Client Satisfaction Index	"Managing client satisfaction"
Process Index	"Improving the way we do business"

Figure 4: Scorecard Indices and their descriptions

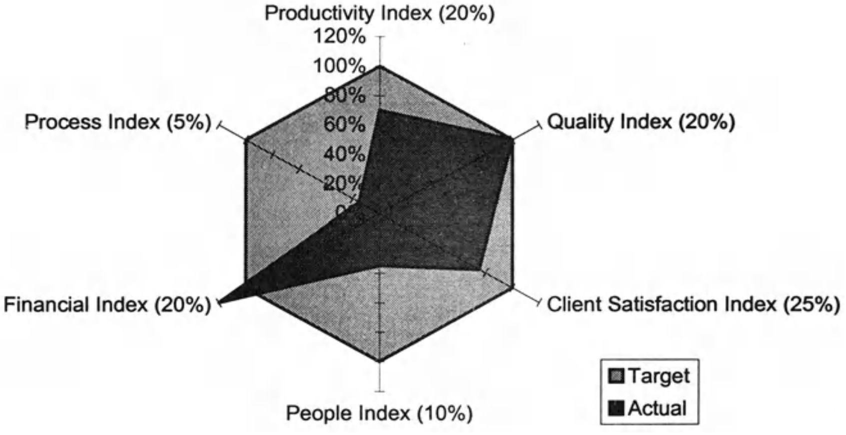


Figure 5: Score-card Indices, target versus actual on a polar chart

Each of the indices will be formed from one or more measures. The Financial index, which forms 25% of the overall score, is shown below as an example.

Financial Index 25%	Weightings		1997	1998	1Q 1998
Measure	S%	Index%	Baseline	Target	Current
Budgeted v actual costs for services	5%	20%	+25%	+15%	-5%
Budgeted v actual costs for development and enhancement projects	5%	20%	+20%	+10%	+4%
Helpdesk cost per user	2.5%	10%	£100	£75	£90
Cost per MIP per user	1.25%	5%	£200	£120	£250
Cost per FP developed	1.25%	5%	£650	£450	£500
Desktop support - average cost per user	2.5%	10%	£50	£45	£50
Total Cost of IS per user	7.5%	20%	£1,900	£1,500	£1,700
% of costs that are variable	2.5%	10%	20%	40%	22%

Figure 6: Components of the Financial Index

The Index% figure represents each particular components contribution towards the financial index score, and the S% figure the contribution towards the overall score-card score. The beauty of the score-card approach is that it enables staff in the organisation to clearly understand what their contribution to the business is, and what the targets for improvements are.

However, there can be some negative aspects to the scorecard approach. For example a senior manager stated to the author that "We cannot afford to spend months carrying out FPA on our systems as Productivity measures only contribute 2% towards the final score-card score". In one organisation, directors set an improvement target of 20 FPs/SM for development projects. This might be fine for those departments achieving less than 20 FPs/SM, but what about those already achieving in excess of 20 FPs/SM?

3.3 Benchmarking

The author initiated a pilot metrics programme and measured two applications, x and y. It transpired that application x had a relatively low productivity of 7.8 FPs/SM compared with application y's productivity of 56.2 FPs/SM. It transpired that application x had been the subject of a very hostile third party audit which revealed a rather lax approach to testing. As a consequence management had initiated improvements in the approach to testing future releases of application x. Unfortunately this knee jerk reaction resulted in an extremely over-bureaucratic approach to testing. Analysis of the effort data revealed that more than 53% of the effort was accounted for by the various phases of testing. There were also two other pertinent factors. Firstly, the project team spent a substantial amount of effort through-out the life-cycle clarifying the requirements. Secondly, the project was substantially de-scoped after requirements, analysis and design had been completed. Therefore some functionality was not delivered, but effort for the requirements, analysis and design of this functionality was still included in the overall effort figure.

3.4 Supplier Management

The author was asked to help verify if the IT organisation was getting good value for money from various external Software Houses who had previously delivered projects for the organisation and were bidding for more work. The cost per FP was used as a comparative measure.

3.5 Cost of Re-write versus Maintenance Costs

Management of a business unit were requesting increased annual resources to maintain and enhance an application, which the directors considered to be excessive and they could see no justification for this. The directors assumed that the application should best be replaced. The author was requested by a third party to carry out a FP analysis to determine the size of the application. It transpired that the application was so large that the author had to spend approximately twenty staff days of effort just to estimate the FP size which was in excess of 40,000 FPs. This prevented the directors from making a serious error, as it would have been expensive and risky to try to write such a large application. As the author had estimated the size business function by business function, the organisation used the analysis to re-engineer the system function by function.

3.6 Risk Management and Schedule Estimate

Senior management requested the author to conduct a reality check on the estimate of 9 elapsed calendar months for the delivery of a business critical application. The author estimated the size of the development (2,600 FPs), and from the resource profile (14 full time equivalent (FTE) staff) and a productivity of 10 FPs/SM obtained a figure of 20 calendar months for the elapsed time. The FP based estimate was ignored by the project manager. The core functionality was delivered after 19 calendar months and the remaining functionality after an additional 3 calendar months. The estimating approach adopted by the author was as follows:

1. A significant amount of requirements gathering, analysis and design work had already been carried out in a previous project that had failed. This provided sufficiently accurate information to estimate the FP size as 2,600 FPs.
2. Systems developed of this size tend to have relatively low productivity rates with an average of around 5 FPs/SM. However, with the knowledge and deliverables gained from the previous failed attempt, and the fact that some of the team have worked together before, this figure was doubled to 10 FPs/SM.
3. This gives a figure of 260 staff months of effort, adjusted to 325 staff months assuming 80% utilisation of staff.
4. 325 staff months of effort with 14 FTE project staff gives 23 calendar months for the schedule.
5. Reducing this by 15% to take into consideration that a lot of the up front requirements and design work has been done in the previous failed attempt gives 20 calendar months for the schedule.

Further evidence to support the fact that the project manager's schedule was too aggressive was obtained from data supplied by the SPR database [8]. The average elapsed time for a project of this size is about 34 calendar months. The SPR database provides the following information for projects around the 1,000 and 10,000 FPs size. The figures for the 3,000 FP size have been obtained by linear extrapolation from the 1,000 and 10,000 FP data.

Size	Minimum	Average	Maximum
1 FP	0.06	0.16	0.40
10 FPS	0.35	1.07	2.35
100 FPs	3.60	10.00	19.00
1,000 FPs	12.24	27.20	43.52
3,000 FPS	15.10	32.22	52.66
10,000 FPS	24.90	49.80	84.66
100,000 FPs	44.28	73.80	132.84
Average	14.24	27.01	47.13

Figure 7: Schedule in calendar months (Source Jones [8], reproduced with permission)

Size	Early	On-time	Delayed	Cancelled
1 FP	14.68%	83.16%	1.92%	0.25%
10 FPs	11.08%	81.25%	5.67%	2.00%
100 FPs	6.06%	74.77%	11.83%	7.33%
1,000 FPs	1.24%	60.76%	17.67%	20.33%
10,000 FPs	0.14%	28.03%	23.83%	48.00%
100,000 FPs	0.00%	13.67%	21.33%	65.00%
Average	5.53%	56.94%	13.71%	23.82%

Figure 8: Probability of selected schedule outcomes (Source Jones [8], reproduced with permission)

4.0 Conclusion

My own research based on the Times top 1000 companies in the UK [9] into this area has revealed that senior managers and directors, as well as other staff lack sufficient training to make effective and informed decisions based upon the findings of their metrics programmes.

Senior managers and Directors of IT organisations must be trained in the science of measurement. They must also understand that introducing measurements will have an impact upon behaviour and attitude. Some behaviours will be positive, others negative. They will need to assess the potential impacts of the measurement programme and be ready to manage any negative behaviours. For example, only providing data for 'good' projects, effort being under-stated for projects and refusing to provide data due to 'pressure of work or commitments'. The greatest risk to any measurement programme is the senior manager or IT director who is not trained to understand how to gather, interpret and use metrics properly.

References

1. Fenton N, Pfleeger S, Software Metrics A Rigorous & Practical Approach, International Thomson Computer Press, London, Second Edition, 1996 ,p511
2. Rubin H, Developing the New IT Scorecard conference, The Copthorne Tara Hotel, London, 6-7 February 1996
3. Jones Capers, Applied Software Measurement, 2nd edition, McGraw Hill, 1996
4. Kemerer Chris F, Reliability of Function Point Measurement: A Field Experiment, MIT Sloan School Working paper 3192-90-MSA, January 1991
5. Ashley N, "Measurement as a Powerful Software Management Tool", McGraw-Hill, 1995
6. Jones Capers; Applied Software Measurement, 1st edition, McGraw Hill, 1991
7. Boehm and Jones in Corbi Thomas, Understanding and Improving Software Development Productivity, IBM Research, order number ZZ28-7068
8. Jones Capers, Patterns of Software Systems Failure and Success, International Thomson Computer Press, London, 1996
9. Banga KS, Tamber HS, An Investigation into the Training and Use of Measurement in Software Engineering, unpublished survey of Times Top 1000 companies and UK Universities, 1997

A Quality Software Process for Rapid Application Development

Gerry Coleman, Centre for Software Engineering
And
Renaat Verbruggen, Dublin City University.

Abstract

Software organisations can significantly improve the quality of their output if they have a defined and documented software process, together with the appropriate techniques and tools to measure its effectiveness. Without a defined process it is impossible to measure success or focus on how development capability can be enhanced. To date, a number of software process improvement frameworks have been developed and implemented. However, most of these models have been targeted at large-scale producers. Furthermore, they have applied to companies who use traditional development techniques. Smaller companies and those operating in development areas where speed of delivery is paramount have not, as yet, had process improvement paradigms available for adoption.

This study examined the software process in a small company and emerged with the recommendation of the use of the Dynamic Systems Development Method (DSDM) and the Personal Software Process (PSP) for achieving software process improvement.

1. Introduction

In order to improve its software capability each organisation must have a defined and documented software process. Defining and documenting the software process allows companies to measure current performance, identify areas of weakness and initiate improvement actions. In 1988 the Software Engineering Institute (SEI) developed a Maturity Framework to appraise the maturity of software processes within companies [1]. This model indicates that the more mature the development process of the organisation, the more capable it is of developing high-quality software. The Maturity Framework included process assessment methods, software capability evaluation, and a maturity questionnaire. After extensive work in this area the SEI evolved the Maturity Framework into the Capability Maturity Model (CMM) [2]. The CMM is based on the knowledge acquired from studies using the Maturity Framework and specifies recommended practices in the particular areas that have been shown to enhance software development and maintenance capability.

C. Hawkins et al. (eds.), *Software Quality Management VI*

The CMM, however, is primarily targeted at large organisations. Small companies also face difficulties in software development and defining appropriate process models. This study examined the problems facing small software companies and proposes using both the Dynamic Systems Development Method (DSDM) [3] and the Personal Software Process (PSP) [4] as a means to process improvement within small companies.

2. Rapid Application Development (RAD) and the Dynamic Systems Development Method (DSDM)

2.1 Rapid Application Development (RAD)

The term Rapid Application Development or RAD is taken to relate to projects based around tight timescales, which use prototyping and combine high-level development tools and techniques [5].

Proponents of RAD claim that it increases productivity, reduces delivery time and gains high usage because of the extent of user involvement in the development [6, 7, 8, 9]. However RAD projects can fail because the following are not properly addressed:

- ❖ Picking the Right Team
- ❖ Management and Customer Support of the project
- ❖ the Methodologies used [10].

Analysis of the material published on RAD reveals the advantages and disadvantages listed in **Table 2.1**.

TABLE 2.1 Advantages and Disadvantages of RAD

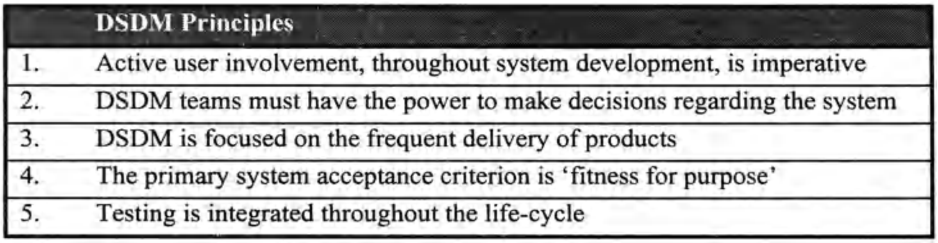
Advantages	*Disadvantages*
Ease of Implementation	Speed of development may result in a poorly designed product
Improved User Satisfaction	Need more experienced development staff
Shorter time-to market	Strong project management and control required

To capitalise on the benefits listed above and to address the potential problems in using RAD, the Dynamic Systems Development Method (DSDM) Consortium was established.

2.2 Dynamic Systems Development Method - A RAD Standard

The Dynamic Systems Development Method (DSDM) was created in February 1995. The objective was to create a quality-centred method within which RAD techniques could be used. DSDM uses prototyping techniques to ensure the frequent delivery of software products during development. These products are delivered within fixed timescales known as 'timeboxes'. The principles on which the method is based are listed in Table 2.2.

TABLE 2.2 – Principles of DSDM



	DSDM Principles
1.	Active user involvement, throughout system development, is imperative
2.	DSDM teams must have the power to make decisions regarding the system
3.	DSDM is focused on the frequent delivery of products
4.	The primary system acceptance criterion is 'fitness for purpose'
5.	Testing is integrated throughout the life-cycle

2.3 The DSDM Life-Cycle

The development life-cycle is divided into five phases:

- Feasibility Study
- Business Study
- Functional Model Iteration
- Design and Build Iteration
- Implementation.

An overview of the life-cycle appears in **Figure 2.1**.

The first phase, the Feasibility Study, determines the feasibility of the project and its suitability for development using DSDM. The Business Study defines the high-level functionality and the affected business areas.

These are then baselined as the high-level requirements together with the primary non-functional requirements. The main part of the development is contained within the two iterative prototyping cycles.

The objective of the Functional Model Iteration is on eliciting requirements while the emphasis in the Design and Build Iteration is on ensuring that the prototypes meet pre-defined quality criteria. The final phase, the implementation phase, is the handover to users, which will normally be accompanied by a project review.

The RAD Process in DSDM

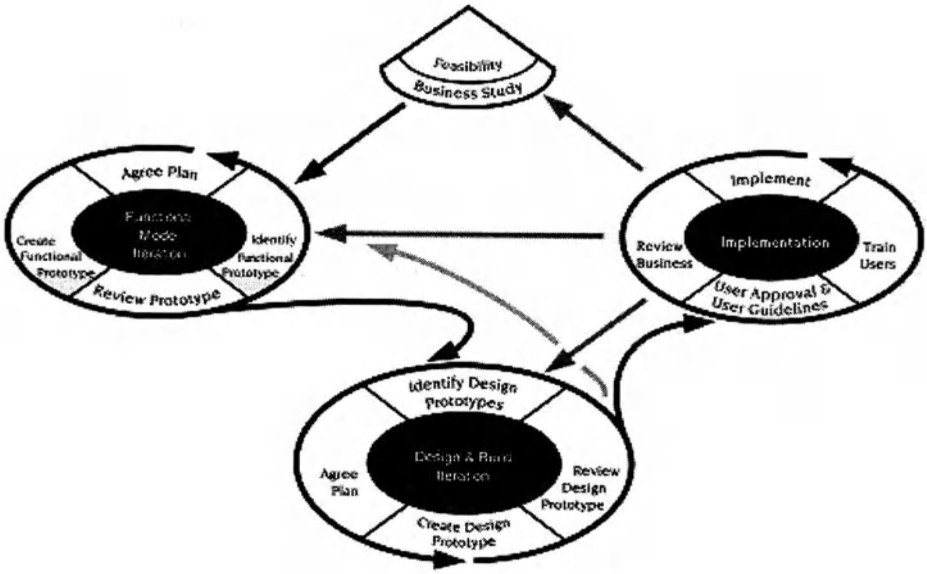


Figure 2.1 - DSDM Life-Cycle

2.4 DSDM - A Way Forward for RAD

Given that there are still fears that RAD produces poor quality, unmaintainable code, there is a definite need for a methodology which will allow quality to be incorporated into RAD projects. DSDM suggests ensuring quality products through:

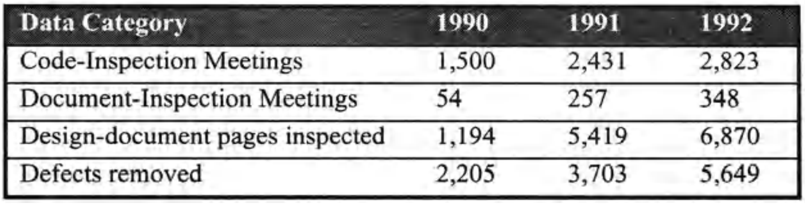
- Inspections, Reviews and Walkthroughs
- Demonstrations to user groups and
- Testing.

While there is a comprehensive account, in the DSDM documentation, of how testing should be conducted at every stage of the development process very little space is devoted to the use of alternative measures of assuring the quality of the developed product. Though the consortium admit that, 'testing can never prove that a system works', limited detail is provided in the documentation as to how alternative quality mechanisms, such as, inspections and reviews, can be used.

A large body of work exists [including 11, 12, 13, 14, 15] which illustrates how inspections and reviews are superior to testing at discovering errors.

The benefits inspections have had in one large company are shown in **Table 2.3.**

TABLE 2.3 - Summary of Inspection Data 1990 to 1992 (from Weller [13])



Data Category	1990	1991	1992
Code-Inspection Meetings	1,500	2,431	2,823
Document-Inspection Meetings	54	257	348
Design-document pages inspected	1,194	5,419	6,870
Defects removed	2,205	3,703	5,649

Backed up by this evidence, it is essential that inspections and reviews form an integral part of DSDM.

Though the consortium state that a formal quality inspection is necessary on the Business Area Definition and the Prioritised Functions, if inspections are carried out only on these areas then testing is being used as the sole mechanism to find defects in the subsequent coding phases. The Functional Prototypes and the Design Prototypes should also be subject to code inspections to ensure maximum defect detection and conformance to requirements.

While the quality approaches in DSDM mirror the best practices in traditional software development there is a greater emphasis on the use of software tools to support the quality process and ensure development momentum. Tools such as those that provide 'capture/playback' testing facilities are imperative as regression testing will occur as new system increments are released.

Finally, the DSDM authors state that every DSDM project should have an accompanying Quality Plan that states how quality and standards are to be enforced.

3. Process Improvement using the Personal Software Process (PSP)

3.1 Personal Software Process (PSP)

The Personal Software Process (PSP) is an attempt to scale down current software quality and assessment practices, for the improvement of individual software developers [4]. It is essentially a bottom-up approach where individuals manage and assess their own work and, as such, is of particular interest to small software houses where tailoring large-scale practices can cause difficulties.

The PSP is essentially a framework of forms, guidelines and procedures, which in effect produce a defined process. Using the PSP data gathering mechanisms provides historical data which helps you measure your performance, your work patterns and practices. By examining these factors Humphrey believes the developer can:

- Make better plans and more accurate estimates
- Improve productivity
- Improve product quality.

3.2 PSP Improvement Phases

The Evolution of the various PSP phases is illustrated in **Figure 3.1.**

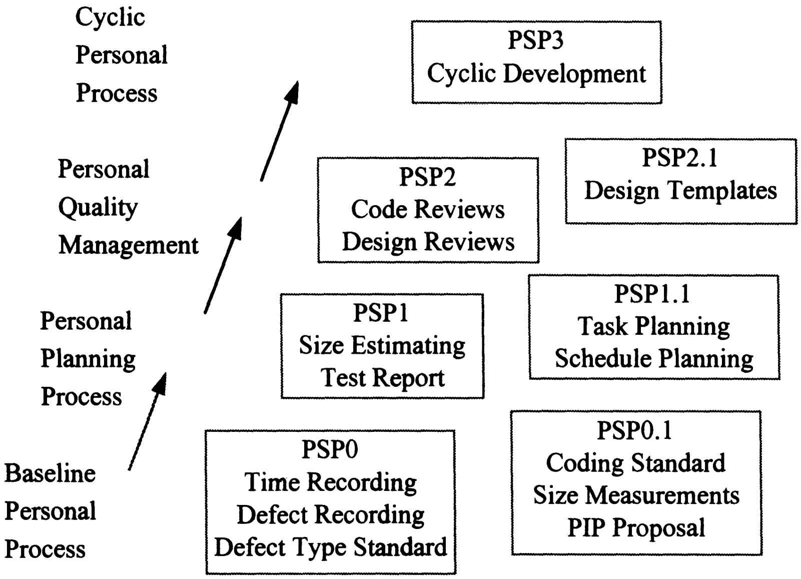


Figure 3.1 - The PSP Evolution

3.3 PSP0 (The Baseline Personal Process)

The principal objective of PSP0 is to provide a framework for gathering your own initial process data.

PSP0.1 extends PSP0 by the inclusion of additional planning and size measurement details. Size can be measured by counting Lines of Code (LOC), Function Points (FPs), Objects or some other suitable unit.

3.4 PSP1 (Planning, Scheduling and Estimation)

In order to assist with size estimation, Humphrey proposes the use of the **PROBE** (***PRO****xy-**B**ased **E**stimating*) method. A ***proxy*** is a substitute or stand-in and in this instance the proxy is used to relate product size to the functions the estimator can visualise and describe. Examples of proxies include objects, screens, files, scripts or function points. PSP1.1 also assists with task and schedule planning.

3.5 PSP2 (Software Quality Management)

PSP2 and 2.1 uses reviews and verification methods as techniques to ensure process and product quality.

3.6 PSP3 (Cyclic Development)

PSP3 proposes decomposing large systems to allow PSP techniques to be used effectively.

3.7 Personal Software Process - An Analysis

Some of the disciplines advocated in the PSP have already been implemented within many companies. Inspections, code coverage, and testing have been used to uncover errors. The cost of fixing a defect when software is in use compared to fixing it if found during coding ($1000 vs. $1) can be exorbitant [16]. Therefore, using code reviews, both for original code and, importantly, for subsequently amended code can result in large cost savings.

The quality of the software product is an issue facing the majority of companies. Most have tackled it by concentrating on the quality of the software process believing that a good product consistently emerging from an ad-hoc process is a remote possibility. This study has already shown how software inspections and reviews have helped a number of companies to find and fix errors earlier in the development life-cycle, when it is cheaper to do so. It has also been shown how, through using inspections, some companies have witnessed productivity increases with software being released earlier and significantly less rework.

Also software metrics are being increasingly used as a way of measuring and controlling the software process. Organisations have found that in the absence of metrics they are unaware of the state of their own process. Collecting even basic defect metrics has, at least, allowed them a preliminary evaluation of their own software capability.

From the point of view of a small software development company the PSP is particularly attractive. As with DSDM it provides a defined and documented software process. Also, small companies will inevitably have small development teams and this will assist greatly with the implementation of PSP. Even one person in a small team using PSP can have a noticeably greater impact on the resultant product quality than one person in a large team. There is also the potential for such a person to act as a 'process champion' with other team members adopting the approaches.

4. Assessment of the Software Process within a Small Company

Without real data on how small companies operate, it is difficult to determine which process models are appropriate and how they should be implemented. The purpose of this section is to examine how software is developed in a small software company.

This was not intended as a software capability assessment but as a first step to gather information about how the company's software process functioned. The company examination is divided into two sections. Firstly, a questionnaire was used to obtain general information on the company's processes. Secondly, having processed the questionnaire responses, a detailed evaluation of the company was then undertaken. This is covered in section 5.

4.1 Questionnaire Findings

The company was assessed under twelve headings.

The figures involved represent the percentage achievement of target.

Target is the carrying out of all the activities, represented under each heading, at all times.

Target is 100%.

The questionnaire results are represented in **Figure 4.1**.

The results show some fundamental process weaknesses, so to get a more detailed picture of how the company develops software a detailed evaluation was then conducted.

5. Evaluation of the Software Process within a Small Software Company

5.1 Evaluation Findings

The evaluation of the organisation's development process highlighted some evident strengths. These included the fact that users were heavily involved in systems development, up-to-date tools were being used very effectively and developers were highly-skilled.

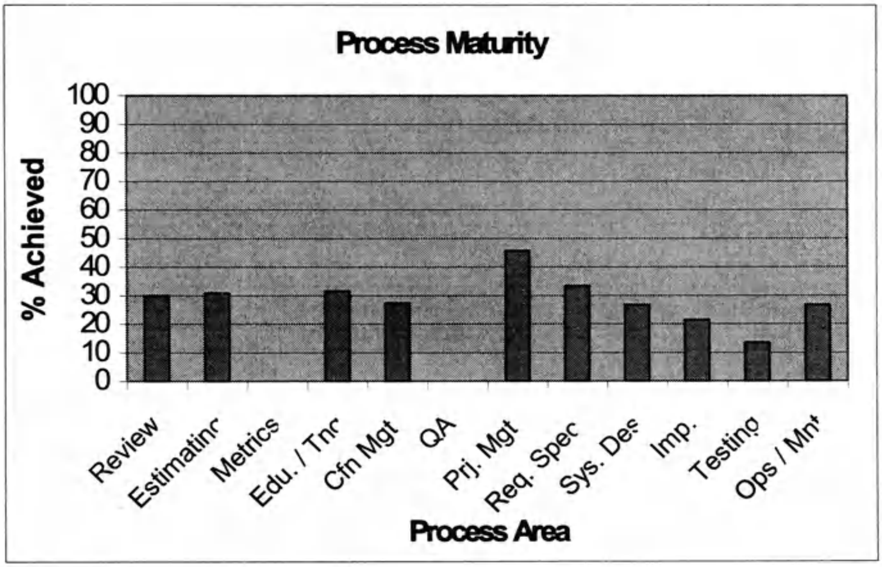


Figure 4.1 Company Process Maturity Chart

However, more fundamental weaknesses were also apparent. Apart from the absence of a defined software development process the company lacks formality and standards in the areas outlined in **Table 5.1:**

TABLE 5.1 Company Process Weaknesses in Standardisation and Procedures

Fundamental Development Process Weaknesses	
1.	No standard User Requirements Document
2.	No standard Design Document
3.	No programming standards exist
4.	Programmers are not required to produce Unit Test Plans
5.	No formal independent testing of modules
6.	No formal documentation of errors found during acceptance testing
7.	No recording of live fault reports and change requests from users

6. A Software Process for RAD

The purpose of this section is to propose how a combination of DSDM and PSP disciplines can be used as a solution to the software process problems highlighted in sections 4 and 5. Throughout this section, the phrase 'the company' refers to the company assessed and evaluated in sections 4 and 5.

6.1 Using DSDM within the Company

There are two questions which need to be answered:

- Is the company suited to the introduction of DSDM?
- Are the projects to be developed suitable for use with DSDM?

The points which favour the use of DSDM within the company are documented in **Table 6.1**.

TABLE 6.1 - Factors which favour DSDM usage within company

Factors Favourable towards DSDM Usage
1. Highly-skilled Development Staff
2. Company application area (Multimedia) already includes major User Involvement in Development
3. Prototyping techniques are currently employed
4. Expertise in latest RAD and software tools
5. Senior Management are committed to improving quality
6. Desire to get products onto the market more quickly

6.2 Using PSP within the Company

The factors which favour the introduction of PSP are outlined in **Table 6.2**.

Table 6.2 - Factors which favour PSP usage within company

Factors Favourable towards PSP Usage
1. Commitment to quality
2. Desire for documented process and standardisation
3. Desire to commence metrics gathering
4. Skilled, enthusiastic developers
5. Small development staff complement

6.3 A Quality Software Process for RAD

While DSDM offers a ready-made life-cycle for RAD it is weak on the application of quality measurement techniques and metrics collection. PSP, conversely, offers process improvement at an individual level but is not RAD-oriented. Taken together DSDM provides the life-cycle and framework which can be used by RAD teams and PSP offers the necessary quality control mechanisms.

6.3.1 Combining DSDM and PSP3

PSP3 offers the closest match with the objectives and framework of DSDM.

The Requirements and Planning stage, of PSP3, carries out similar functions to DSDM's Business Study and the Cyclic Development section closely matches the Design and Build Iteration in DSDM. Where DSDM scores over the PSP, for the type of applications used by the company in this study, is the increased emphasis on iteration particularly in the Functional Model stage.

However, PSP3 includes unit and integration test within each of its development cycles. This is particularly important as each increment is adding extra functionality to the previous increment. If there are residual defects in the initial increment then these may cause a 'ripple' effect in subsequent increments. Furthermore, subsequent increments must also find these defects. PSP3 insists on the use of reviews to prevent this. Using reviews at each of the DSDM prototyping phases will ensure that 'clean' versions of software are used as input to subsequent system increments.

6.3.2 Using Proxies

The DSDM consortium contend it is easier to calculate how much can be done by a certain time than to calculate how long it takes to do something; thereby promoting the use of timeboxes within which given portions of functionality can be delivered. The PSP suggests using Proxies as a size estimating technique. If the proposed data collection and recording mechanisms are used, you will soon have historical data from which you can determine how many LOC or Function Points (FPs) can be developed within the timebox. In the absence of historical data and size measurements, the company is at risk of overestimating development time and therefore losing potential customers or substantially underestimating development time with the consequent failure to meet deadlines, reduced quality end products and/or substantial overtime requirements or extra staff complement. The use of proxies will help this company refine its estimates.

6.3.3 Testing

In the section of the DSDM documentation devoted to testing, the consortium state that, though testing is burdened by time and resource constraints, no amount of testing would locate all errors.

The consortium also state that during testing 'confidence is derived from finding errors which are then fixed'. Users who have spent many excessive hours in acceptance testing would balk at this statement. Continually finding errors in products at this stage substantially reduces their confidence in the system, as they have no way of knowing how many errors they are not finding!

Without proper reviews and inspections, prior to product handover to the user, there is potential for the user to receive error-laden code. This could certainly reduce the user's confidence in the system and generate substantial rework. In such an environment, development schedules would not be met and development time would be lengthened thus invalidating the use of RAD.

An improved procedure would be for each individual developer to conduct a design and code review with testing then executed by a technical peer. This will ensure that the product the user receives will be more error-free. This would increase user confidence in the product and allow them to concentrate their testing efforts on the major areas of system functionality.

Any moves towards improved testing procedures within the company should be accompanied by a commensurate effort to improve inspection and review techniques. The DSDM Consortium recommend the use of static code analysers since they do 'a degree of code inspection almost for free'. However, until the company is wholly satisfied with the specific code analysis tools, available for use with its development environment, it should concentrate on a manual inspection/review approach for code.

6.3.4 A Quality Plan

It is stated that every DSDM project should have a quality plan that outlines how quality will be controlled and standards applied. The proposed quality software process for RAD will have a quality plan and could include much of the documentation provided by the PSP.

For example, the PSP process scripts for each level of the PSP should be included in a Quality Plan.

Having a process script which outlines the Entry and Exit criteria for a phase with a detailed outline of the steps to be undertaken during that phase is a very useful quality control technique. The process scripts are accompanied by planning, development and postmortem scripts for each phase and these again ensure quality coverage of the process. Including these documents in a quality plan introduces the standards and control elements demanded by DSDM.

Another quality control mechanism, which could be adapted from the PSP for use in the new environment, is the checklist. Design and code review checklists, which could be established for each language used by the organisation, and any documentation used with software inspections should form part of the Plan. The PROBE estimating script, introduced in PSP1, could also be included in the Plan.

Another PSP document which would be part of the quality plan is the Process Improvement Proposal (PIP), which is part of PSP0.1. The PIP provides a means of documenting process shortcomings and suggested solutions. PIP copies should be retained in the quality plan.

One way of assessing the effectiveness of the quality plan is to measure the process itself. The PSP provides quality measures for use in PSP projects. However, some new measures are appropriate for use in the quality software process for RAD.

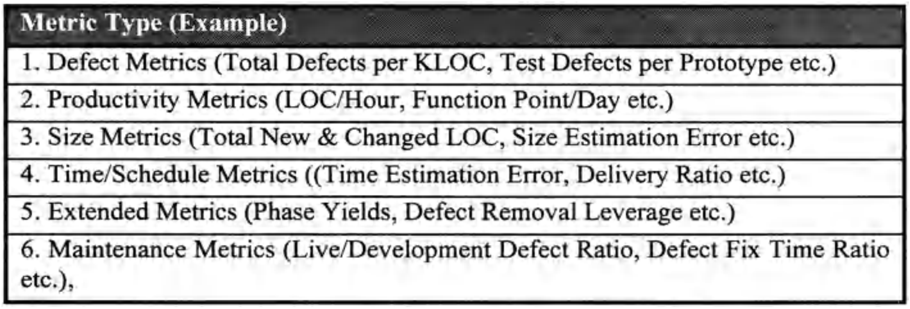
6.3.5 Metrics

At present, the DSDM consortium is not recommending any specific approach towards collecting metrics. Also, what is stated, in the DSDM Manual, is that 'during system testing, a count of detected errors should be kept with the aim of improving this as time goes on'. If a process is to be successful, and continually improve, it is too late to start collecting error metrics at the system testing stage.

There are several reasons for this. Firstly, it may be very difficult to determine the exact cause of the error, as the defect, which gives rise to the error, may have been introduced during any previous timebox or prototyping phase. Secondly, no record will be available of the time it has taken to fix errors (which have been removed prior to system testing) and during which activity they were removed. Thirdly, there will be no picture available of how effective the interim testing is i.e. testing of timebox elements, prototypes etc. Also, DSDM proposes the analysis of errors and their causes. Again, without collecting error data as the project progresses, identifying error causes will be a complex and imprecise activity.

A list of the metrics proposed for use within the company appears in **Table 6.3**.

TABLE 6.3 - Metrics which could be introduced effectively into the company



Metric Type (Example)
1. Defect Metrics (Total Defects per KLOC, Test Defects per Prototype etc.)
2. Productivity Metrics (LOC/Hour, Function Point/Day etc.)
3. Size Metrics (Total New & Changed LOC, Size Estimation Error etc.)
4. Time/Schedule Metrics ((Time Estimation Error, Delivery Ratio etc.)
5. Extended Metrics (Phase Yields, Defect Removal Leverage etc.)
6. Maintenance Metrics (Live/Development Defect Ratio, Defect Fix Time Ratio etc.),

It is not recommended that all of these metrics should be introduced immediately into a project. It is only through using and collecting metrics that companies or individuals can decide which are the most appropriate and cost-effective to collect.

6.3.5.1 Size and Defect Metrics

Size can be counted at all stages of development and can be based on Lines of Code (LOC) or Function Points (FP) delivered. These figures can be used to determine the estimation errors in code size, i.e. % difference between estimated size and actual size, and can be calculated as follows:

$$\text{Error\%} = 100 * \frac{(\text{Actual Size - Estimated Size})}{\text{Estimated Size}}$$

Size estimation errors, by prototype and by timebox, should also be measured.

Defects will be measured relative to code size. Measuring defects by timebox and by prototype produces the following examples:

$$\text{Total Defects/Size (Timebox)} = \frac{\text{Total defects per Timebox}}{\text{Size per Timebox}}$$

for assessing the effectiveness of the testing process:

$$\text{Test Defects/Size (Prototype)} = \frac{\text{Test defects per Prototype}}{\text{Size per Prototype}}$$

A defect database should be maintained containing information about the defects injected and removed during development phases. The database should contain fields for Defect Number, Defect Type, Inject Phase, Remove Phase and Fix Time. A Prototype Identifier could also be recorded. Additionally, new fields, such as Timebox Number and Object/Function/Method Number and Type (e.g. I/O, Interface etc.) could be included. This extra data will assist in future defect analysis.

6.3.5.2 Productivity Metrics

The standard productivity measures relate to code amendments and additions.

If we treat **Additions** as either new and changed lines of code (NLOC) or new and changed function points (NFP) and **Time Units** of hours in relation to NLOC and days in relation to NFP then **Productivity** can be measured as:

$$\text{Productivity} = \frac{\text{Total Additions}}{\text{Development Time Unit}}$$

It is also recommended to measure productivity by timebox and by prototype. However, it should be noted that using RAD tools where a lot of screen design is completed without the requirement to write code, productivity figures, if based on lines of code, can look low.

Similarly, if there is significant code reuse then again productivity figures may be underestimated. The above two elements should be factored into any productivity estimates and calculations.

6.3.5.3 Time/Schedule Metrics

For future estimates of development time or deliverable capability per timebox it is necessary to collect time measures. These time measures can be used to determine how much time was spent in each of the particular development phases and should be counted in the most appropriate units e.g. hours, days, weeks etc.. If the productivity measures are to be accurate, then time must also be recorded per prototype and per timebox. Time estimation error, i.e. % difference between estimated time and actual time, can be calculated as follows:

$$\text{Error\%} = 100 * \frac{(\text{Actual Time - Estimated Time})}{\text{Estimated Time}}$$

However, using a formula similar to the one for error estimation can assist in measuring the closeness between predicted timebox deliverables and actual timebox deliverables. As the prime objective in DSDM is 'building the right system' this metric will serve as both a quality and productivity measure for the development.

Assuming this measure is termed the Delivery Ratio (DR) it can be calculated thus:

$$\text{DR (Total Development)} = 100 * \frac{\text{Delivered Functionality}}{\text{Planned Functionality}}$$

This should also be calculated by prototype and timebox. Also, the measure could be used at the end of any iterative phase or timebox to determine the ratio of delivered functionality to date:

$$\text{DR (Milestone A)} = 100 * \frac{(\text{Delivered Functionality to Milestone A})}{(\text{Planned Functionality to Milestone A})}$$

6.3.5.4 Extended Measures

With the basic metrics in place, some derived measures can also be generated.

The Yield of a phase is the percentage of defects removed in a phase over the total number of defects removed and remaining in the product.

For example, a prototype yield could be:

$$\text{Yield (Prototype)} = \frac{100 * \text{(Defects found in Prototype)}}{\text{(Defects Found in Prototype + Escaping from the Prototype)}}$$

This measure will be useful in determining the quality of the defect detection mechanisms used.

The Defect Removal Leverage (DRL) provides a measure of the effectiveness of different defect removal methods. The DRL is the ratio of the defects removed per hour in any two phases and is particularly useful in comparing say a review phase with a test phase. These approaches could again be profitably employed in DSDM by measuring defect removal rates in prototypes and timeboxes.

For example, the measures could be calculated for a prototype as:

$$\text{DRL (Code Review for Prototype A)} = \frac{\text{Defects/Hour (Code Review for Prototype A)}}{\text{Defects/Hour (Unit Test for Prototype A)}}$$

High review yields are preferable to high test yields as they show that defects are being found earlier in the process and are thus cheaper to correct.

6.3.6 Maintenance

Projects developed in RAD environments have, in the past, been criticised for being unmaintainable. The quality software process for RAD proposed in this study will improve the maintainability of software through improving quality. To measure maintenance effort post-delivery defects should be collected. This measure, the Live to Development Defect Ratio (LDDR), could be used to determine the ratio of defects found in the live environment to those found in development:

$$\text{LDDR} = 100 * \frac{\text{Defects found in live environment}}{\text{Defects found during development}}$$

Another complementary maintainability measure would be the time spent fixing defects after development, calculated as the average defect fix time post-delivery versus the average fix time prior to delivery. The average defect fix time during development could be calculated thus:

$$\text{Avg. Defect Fix Time (during development)} = \frac{\text{Total defect fix time during development}}{\text{Total number of defects during development}}$$

The average defect fix time post development would be:

$$\text{Avg. Defect Fix Time (post development)} = \frac{\text{Total defect fix time post development}}{\text{Total number of defects post development}}$$

Therefore, the Defect Fix Time Ratio (DFTR) for two phases would be:

$$\text{DFTR (Phase A /Phase B)} = \frac{\text{Average Defect Fix Time (Phase A)}}{\text{Average Defect Fix Time (Phase B)}}$$

For Post-Delivery versus Pre-Delivery this would be:

$$\text{DFTR (Post-delivery / Pre-delivery)} = \frac{\text{Average Defect Fix Time (Post-Delivery)}}{\text{Average Defect Fix Time (Pre-Delivery)}}$$

These metrics are very significant in that they show clearly the time and cost implications of not catching defects early when they are cheaper to fix.

6.3.7 Software Reuse

Both DSDM and PSP state that reusing existing software components will result in productivity and quality improvements. There is no doubt that there are significant productivity benefits to reusing software. There are, however, significant costs associated with reuse. Apart from the extra time it takes to design and develop reusable parts, there is the question of managing and maintaining the reuse library. A DSDM project which possesses tight delivery deadlines may not have the time to develop reusable parts. They may have to be re-engineered later. There is always the possibility though that this re-engineering may then necessitate and trigger other re-engineering requirements within the same project. It is up to the company, implementing the PSP and DSDM to decide what its own policy is on reuse and where the costs are to be borne. Ultimately, from a corporate perspective, there is no benefit to developing reusable components if the procedures and personnel are not in place to manage and control software reuse component libraries.

7 - Conclusions

7.1 Detailed Analysis

The study revealed that implementation of DSDM would provide a road map for development within the company and a means of achieving Capability Maturity Model (CMM) level 2.

DSDM offers the potential for faster delivery of software. To ensure that the software is of sufficient quality requires the quality control and assurance mechanisms, such as checklists, reviews and inspections, associated with PSP to be used. Further benefits of using the PSP with DSDM include:

- PSP3 supports unit and integration testing during each iterative cycle.
- Proxy-based estimating will help with defining timebox elements in DSDM.
- PSP supplies appropriate metric-collection techniques that can be used in DSDM projects.
- The documents and scripts associated with PSP can be used in a Quality Plan.

7.2 Recommendations

7.2.1 Data Recording Assistance

There is no doubt that the detailed form filling required to gather all of the necessary data for process measurement within the PSP will hinder the speed of development in the initial phases. In order to prevent this, organisations using PSP measures within a RAD framework could either simplify the recording approaches or investigate the use of software tools to collect some of the basic data. One approach would be for developers to maintain detailed logs of their time on randomly selected days only [17]. With respect to using software tools to reduce the data collection overhead, there are a number of tools available to count lines of code (LOC). In areas where the user interface is crucial, LOC counting may not be as effective as function point counting. Tools which can count function points after development would be of tremendous benefit in such an environment.

Similarly, tools which assist in defect and time recording would be advantageous.

7.2.2 Inspection/Review Assistance

DSDM recommends the use of automated support for code reviewing and inspection. Particular benefit will be gained if these tools can be adapted for use with the PSP. Ideally then, they could not only highlight code errors but by reference to, perhaps, the PSP defect standard, they could record the category of defect. These tools, however, should only be considered when the manual review and inspection process is fully understood.

8. References

[1] Humphrey, Watts S. Characterising the Software Process: A Maturity Framework. IEEE Software; March 1988; 73-79.

[2] Paulk, Mark C., Curtis, Bill, Chrissis, Mary Beth & Weber, Charles Capability Maturity Model, Version 1.1. IEEE Software; July 1993; 18-27.

[3] DSDM Consortium, Dynamic Systems Development Method, Version 2.0. November 1995.

[4] Humphrey, Watts S. A Discipline for Software Engineering. Addison Wesley, 1995.

[5] Martin, James Rapid Application Development. Macmillan, 1991.

[6] Gordon, V. Scott & Bieman, James M. Rapid Prototyping: Lessons Learned. IEEE Software; January 1995; 85-94.

[7] Kerr, James & Hunter, Richard Inside RAD - How to build fully functional computer systems in 90 days or less. McGraw Hill, 1994.

[8] Luqi & Royce, Winston Status Report: Computer-Aided Prototyping. IEEE Software; November 1991; 77-81.

[9] Reilly, John P. Does RAD Live Up to the Hype?. IEEE Software; September 1995; 24-26.

[10] Carmel, Erran Does RAD Live Up to the Hype?. IEEE Software; September 1995; 25-26.

[11] Fagan, M.E. Design and Code Inspections to Reduce Errors in Program Development. IBM Systems Journal; 1976; Vol. 15; No. 3; 182-211.

[12] Fagan, M.E. Advances in Software Inspections. IEEE Transactions on Software Engineering; July 1986; Vol. SE-12; No. 7; 744-751.

[13] Weller, Edward F. Lessons from Three Years of Inspection Data. IEEE Software; September 1993; 38-45.

[14] Davis, Alan M. Fifteen Principles of Software Engineering. IEEE Software; November 1994; 94-101.

[15] Ackerman, A. Frank, Buchwald, Lynne S.& Lewski, Frank H. Software Inspections: An Effective Verification Process. IEEE Software; May 1989; 31-36.

[16] Joch, Alan & Sharp, Oliver How Software Doesn't Work: Nine Ways to Make Your Code More Reliable. Byte; December 1995; 49-60.

[17] Shostak, Barry Adapting the Personal Software Process to Industry. Software Engineering Technical Council Newsletter; Winter 1996; Vol. 14; No. 2; 10-12.

An integrated Model for Impact Analysis of Software Change

Henri Basson

Laboratoire Informatique du Littoral
3, rue Louis David , P.B. 719,
62228, Calais cedex France.
E-mail: { basson@loria.fr }

Abstract. This paper presents an integrated model intended to be used for an *a priori* analysis impact of the change of software components issued from different phases of the software life cycle.
The model takes into account different types of links between components, as well as several properties permitting to distinguish between different occurrences of a same relationship type between components. The model permit also to observe components dependencies, and to identify software components concerned by an intended software change.

1 Introduction

The impact analysis of software change is largely admitted as a vital activity to control and manage software evolution as well as to ensure coherence between different representations of software components throughout the software life cycle. The proposed models to describe and handle software components are seemingly not exhaustive[2, 3, 4, 6] such as to take into account various types of links between software components.
The assuranec of coherent description between different representations of components through different phases of the software life cycle, is recognized to be an essential characteristic for software maintainability. It has created the need for integrated representation permitting an *a priori* impact analysis of software change and an exhaustive description of relationship types between software components. In this paper, we propose an integrated representation of components allowing an easier observation of components interdependencies, and better control of software change.

2 Model presentation

To outline the profile of the intended software components representation, we start from a set of requirements we expect to be met by the future model. The requirements specification here is pragmatic based on background in dealing with the needs of information of software change throughout the software life cycle.
As we aim at providing a model allowing an exhaustive impact analysis of software change, according to some background in the area of software evolution, we mention below the main model requirements:

C. Hawkins et al. (eds.), *Software Quality Management VI*

- A complete integrated representation of the software life cycle components.
- A precise identification the structural "coordinates" of each software component within the life cycle phases.
- Representation of different relationship types which may occur between software components, that includes: (i) the representation of different relationship types between a component of a given phase and other concerned components of the other lifecycle phases. (ii) The representation of different relationship types between one component and other components of the same phase.

- The representation of components properties
- The representation of different properties of relationship types occurrences between components.

3 Model dimensions

Successively, we attempt below to meet the roughly mentioned requirements. For structural coordinates the model represents first phase to which the concerned component belongs. Components of the same phase are generally of different "granularity", "category" or "class", then according to some *components classification*, the model represents the category or granular level to which a component belongs. Each component may, in turn, be composed of other components of the same or lower level of classification.

As example, in the coding phase of a project implemented in Ada , the model takes into account four granular levels:

- Modular level in which each PACKAGE, PROCEDURE, FUNCTION or TASK constitutes a modular component.
- Bloc level, which is identified in a component as well-defined set of successive statements, where the execution control flow passes unconditionally from a current statement to the physically succeeding one.
- Statement level where elements are simple or compound statements.
- Symbol level.

A granular level includes generally different sub-categories or sub-levels. The specification of sub-levels is realized using a set of criteria relevant to the software change analysis. For instance in the level Bloc we identify four different sub-levels (declarative bloc, conditional bloc, functional bloc, comment bloc). A change in a declarative bloc has an impact generally different from a change of a comment bloc, etc.

To represent *relational* point of view under which components can be considered, we have taken into account:

- Different types of relationship which may occur between components issued from different phases of the life cycle.
- Different Relationship types between components which belong to the same granular level (horizontal relationship types).
- Different relationship types between components of distinct granular levels (vertical relationship types).

The *qualitative* point of view is represented through attributes permitting to describe precisely different characteristics, pertinent for software change of components of phases, levels, sub-levels, as well as inter-phases, horizontal, and vertical relationships.

4 Model components

We consider in the model the phases which are present in various models of projects execution (Waterfall, V, incremental, spiral, etc.). These phases are such as: requirements specification, functional specification, preliminary design, etc.

The life cycle phases constitute the higher level set of the model components, it is denoted by PS (Phases Set), where:

$PS = \{\Phi_f,\ f = 1 \ldots nf\} = \{RSOS,\ FSOS,\ PDOS,\ DDOS,\ COS,\ ITOS,\ IOS,\ GTOS\}$ denotes the phases set including all components issued from the adopted model execution of development process, we denote by:

- $RSOS$ the requirements specification component set;
- $FSOS$ the functional specification component set;
- $PDOS$ the preliminary design component set;
- $DDOS$ the detailed design component set;
- COS the coding component set;
- $ITOS$ the individual testing component set;
- IOS the integration components set;
- $GTOS$ the global testing component set.

The history of components change is described through two sets:
- $CROR$, the set of change reasons and corresponding managerial and/or technical decisions, and,
- CER, the component set of change carrying out.

Between each phase and (CROR,CER) a set of relationship types are occurred, indicating changed components or components candidate to be changed , why a component was or is to be changed, the corresponding change decision, and description of carried out changes.

Between each couple of successive phases, a set of inter-phases relationships is defined to cover different types of links which may occur between components issued from these phases [fig. 1] As examples of inter-phases relationship types: represents, specifies, is_the_design_of , implements, tests, etc.

4.1 Phase components classification

Since the software change, in a given phase, may concern component of a small or large size, local or global component, etc. the components modeling has had to take into account components specific characteristics. However, components can be distinguished from various standpoints depending on action to be undertaken on components, thus, the model building requires to select a set of specific criteria with which some classification components can be done. Each selected criterion is supposed to be tightly related to the needs of information of software change execution and management. Subsequently, there is no systematic way with which criteria can be selected, the designation is further pragmatic based on experience in software change activities. The components classification will be followed by the definition of intra-relationship types between elements of each distinct subset of components of the same phase (horizontal relationship types), as well as inter-relationship types between elements which belong to distinguished subsets of this phase (these generally correspond to vertical relationship types). The first criterion is component *granularity*, with which we identify distinct granular levels. Inside each one we apply another classification pertinent for software change analysis.

The model formal notations for the coding phase, are given as annex in the end of the paper.

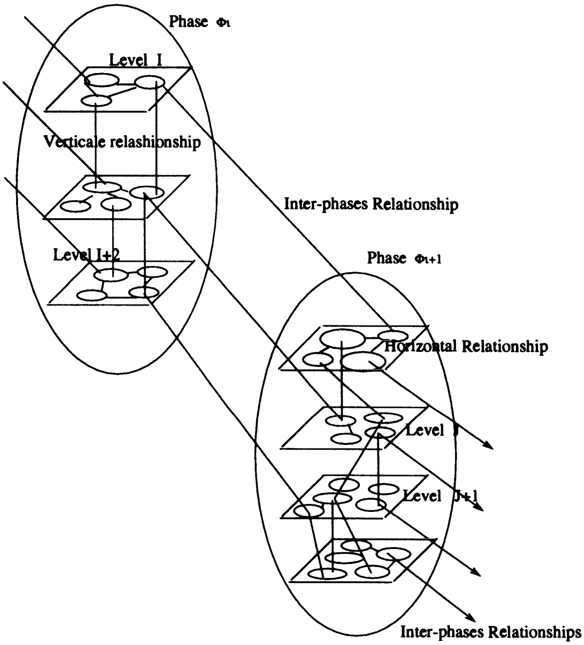


Fig. 1. Direct inter et intra-phases relationships

5 Relationship types example

To illustrate the types of relationship, which may occur between objects, as well as the description of properties permitting to specify precisely their context, we give below two examples, first is various possible types of horizontal relationships between Ada modular components, the second is about vertical relationships between modular component and the sub-level operand included in symbol level.

As example, let us consider the basic horizontal relationship types between modular components of the *first level* of coding, it may include:

$\mathbf{r}(\Phi_\mathbf{f}, \mathbf{1}, \mathbf{1})$: *Calling* ;
$\mathbf{r}(\Phi_\mathbf{f}, \mathbf{1}, \mathbf{2})$: *Reference of common data* ;
$\mathbf{r}(\Phi_\mathbf{f}, \mathbf{1}, \mathbf{2})$: *Reference of common modular components* ;
$\mathbf{r}(\Phi_\mathbf{f}, \mathbf{1}, \mathbf{3})$: *Definition of values of common data* ;
$\mathbf{r}(\Phi_\mathbf{f}, \mathbf{1}, \mathbf{4})$: *Exportation of data* ;
$\mathbf{r}(\Phi_\mathbf{f}, \mathbf{1}, \mathbf{5})$: *Exportation of data types*;
$\mathbf{r}(\Phi_\mathbf{f}, \mathbf{1}, \mathbf{6})$: *Exportation of modular components* ;
$\mathbf{r}(\Phi_\mathbf{f}, \mathbf{1}, \mathbf{7})$: *Importation of modular components* ;
$\mathbf{r}(\Phi_\mathbf{f}, \mathbf{1}, \mathbf{8})$: *Importation of data* ;
$\mathbf{r}(\Phi_\mathbf{f}, \mathbf{1}, \mathbf{9})$: *Importation of data types*;
$\mathbf{r}(\Phi_\mathbf{f}, \mathbf{1}, \mathbf{10})$: *Protection* ;
$\mathbf{r}(\Phi_\mathbf{f}, \mathbf{1}, \mathbf{11})$: *Inclusion* ;
$\mathbf{r}(\Phi_\mathbf{f}, \mathbf{1}, \mathbf{12})$: *Communication* ;
$\mathbf{r}(\Phi_\mathbf{f}, \mathbf{1}, \mathbf{13})$: *Synchronization* ;
$\mathbf{r}(\Phi_\mathbf{f}, \mathbf{1}, \mathbf{14})$: *Instantiation* ;
$\mathbf{r}(\Phi_\mathbf{f}, \mathbf{1}, \mathbf{15})$: *Overloading* ;
$\mathbf{r}(\Phi_\mathbf{f}, \mathbf{1}, \mathbf{16})$: *Renaming* ;
... : .... ;

$\mathbf{r}(\Phi_\mathbf{f}, \mathbf{1}, \mathbf{n})$: ....}

As already mentioned, a set of attributes for will be chosen to insure a large coverage of data requirements of change which may be carried out on modular components.

6 Relationship type and Change Conductivity

Relationship type occurring between components of any couple, may imply, some kind of dependency, reciprocal or not, between components [fig. 2].

A dependency aspect between the components of any given couple, concerns the change of one them or both. When one component of a couple is modified the other could, in somehow, be concerned.
As example, let us consider a couple of software components: C_x and C_y (two procedures of Ada or C application), related by the calling relationship type : C_x calls C_y. Subsequently, in the most of cases, *when C_y is changed then C_x is concerned.* The extent to which C_x is concerned depends on the type of change implemented on C_y.

On the other hand, a change taking place on one of the components, generates an impact of several dimensions, which reflect the impact effect, considered from different points of view. The impact dimensions includes *functional*, *qualitative*, *behavioral* points of view, under which the impact can be considered, in order to understand the consequences of impact change.

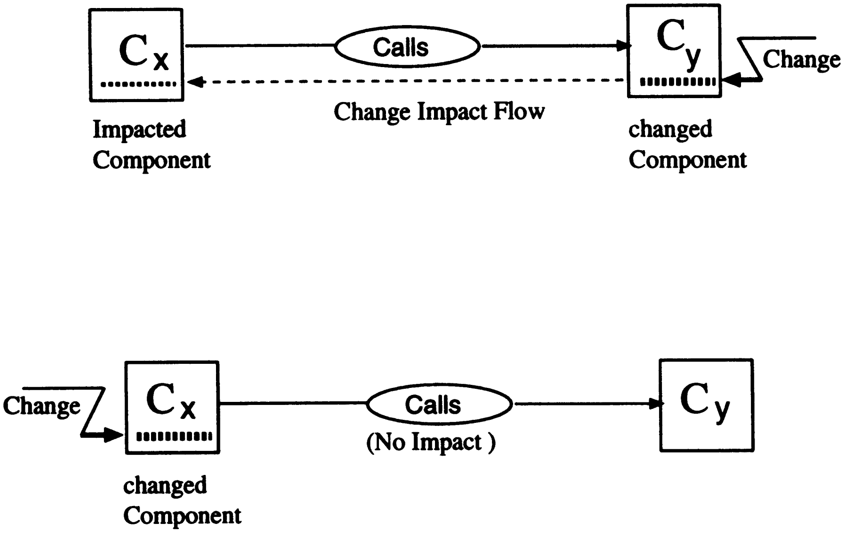


Fig. 2. Conductivity of call relationship type for change impact

7 Attributes identification

For the attributes identification corresponding to properties qualifying objects and occurring relationships. for this objective, we start from a *representative* set of software processes categories, for each one, and by top-down refinement, we proceed to identify necessary operands values permitting to evaluate expressions frequently requested by concerned software process.

In order to provide operands values, a set of attributes will be integrated in the model elements. So the process of attributes identification detailed in is pragmatic based on a background in software evolution needs and data design ??.

The identified necessary attributes will be appended on concerned:

- sofware life cycle phases;
- granular levels of a given phase;
- component types of a given level;
- sub-levels of a given level;
- component types of a given sub-level;

- types of inter-phases relationships;
- horizontal types of relationship;
- vertical types of relationship.

8 Example of associated qualification of the calling relationship

Each calling between two modular components can be characterized by a set of attributes permitting to qualify it precisely. So the k^{th} calling between two components C_i and C_j is characterized by the values of the chosen attributes such as:

- *calling_location*(k) which designates the kth location at which C_i may call C_j
- *calling_condition*(k) designates the condition of calling at the kth location.
- *probable_number_of_calling*(k) That designates the total number of calling at the kth location.
- *nesting_level_of_calling*(k)
- etc.

9 Example of modular vertical relationships

As example of vertical relationship types between components which belong to two different granular levels, we consider relationships between modular level components $LV(\Phi_f, 1)$ and those of operand sub-level $SLV(\Phi_f, 4, 1)$.
The set $V(\Phi_f, 1, 4) = \{v(\Phi_f, 1, 4, k), k = 1 \ldots ln(\Phi_f, 1, 4)\}$, may be instantiated as follows:

$\mathbf{v(1,4,1)(C(1,i),C(4,j))} \longleftrightarrow$ *operand* $C(4, j)$ *is declared in* $C(1, i)$

$\mathbf{v(1,4,2)(C(1,i),C(4,j))} \longleftrightarrow$ *operand* $C(4, j)$ *is initialized in* $C(1, i)$

$\mathbf{v(1,4,3)(C(1,i),C(4,j))} \longleftrightarrow$ *operand* $C(4, j)$ *is read in* $C(1, i)$

$\mathbf{v(1,4,4)(C(1,i),C(4,j))} \longleftrightarrow$ *the current value of* $C(4, j)$ *is modified in* $C(1, i)$

$\mathbf{v(1,4,5)(C(1,i),C(4,j))} \longleftrightarrow$ *operand* $C(4, j)$ *is used in* $C(1, i)$

$\mathbf{v(1,4,6)(C(1,i),C(4,j))} \longleftrightarrow$ *operand* $C(4, j)$ *is protected in* $C(1, i)$

$\mathbf{v(1,4,7)(C(1,i),C(4,j))} \longleftrightarrow$ *operand* $C(4, j)$ *is exported from* $C(1, i)$

$\mathbf{v(1,4,8)(C(1,i),C(4,j))} \longleftrightarrow$ *operand* $C(4, j)$ *is imported by* $C(1, i)$

$\mathbf{v(1,4,9)(C(1,i),C(4,j))} \longleftrightarrow$ *operand* $C(4, j)$ *is visible by* $C(1, i)$

$\mathbf{v(1,4,10)(C(1,i),C(4,j))} \longleftrightarrow \textit{operand } C(4,j) \textit{ is displayed by } C(1,i)$

$\mathbf{v(1,4,10)(C(1,i),C(4,j))} \longleftrightarrow \textit{operand } C(4,j) \textit{ is stored by } C(1,i)$

$\mathbf{v(1,4,11)(C(1,i),C(4,j))} \longleftrightarrow \textit{operand } C(4,j) \textit{ is actual parameter in a calling of } C(1,i)$

As horizontal relationship type, can be used individually or with other types to identify impact flow of software change [fig. 3].

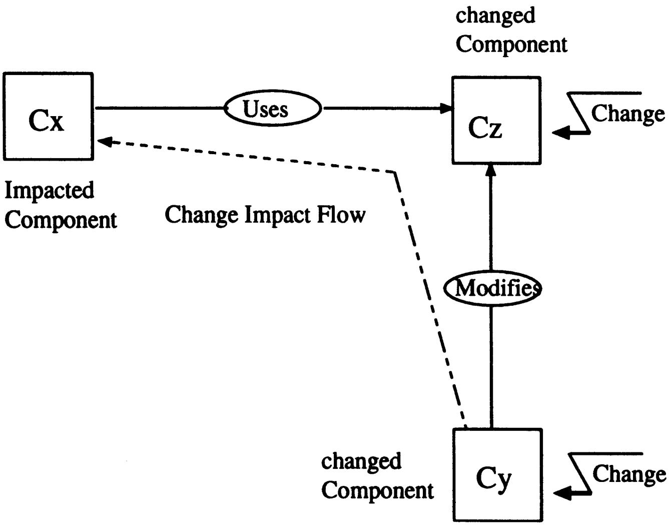


Fig. 3. Flow impact change through modification and use relationship type

10 Impact Flow Graph

Considering the impact flow sense corresponding to each individual relationship type, we can deduce for the instantiated model of the current application the change Impact Flow. The procedure is described in detail in [5]. It is based on the exam of the "conductivity" of each relationship type for the impact change propagation. This conductivity, which can be conditional depending on cases, conducts a full or partial impact.

11 Concluding remarks

The proposed model covers all type of relationships which may occur between software components. For impact analysis of software change, we associate to the application multi-graph, a multi-graph of Change impact flow, which allows, when a component is modified, to identify the other concerned components, throughout the different phases of the software life cycle.

As result of the number of considered types of relationship, the elaborated model is comparatively a large one. In fact, that reflects the number of impact flow paths across them, effects of change, can propagate. Generally, it remains a very difficult task to the maintenance team to identify completely impact flow of software change without assisting tool. We have implemented a tool validating the model for the two phases of preliminary Design and Coding, presented in [1].

Annex: Model elements and notations for coding phase

We give below the representation of one *macro-component* of the phases set PS, the COS or the coding object set for which we define its structure and elements relationships. The process can be similarly applied to the other development phases. for COS elements partition. of coding objects Let $COS = \Phi_f = \{LV(\Phi_f, i),\ i = 1 \ldots ln\}$ be the set of $ln(\Phi_f, i)$ levels To each level $LV(\Phi_f, i)$ corresponds the couple $(SC(\Phi_f, i), R(\Phi_f, i))$, where:
$SC(\Phi_f, i) = \{C(\Phi_f, i, j),\ i = 1 \ldots\ ln(\Phi_f, i),\ j = 1\ \ldots\ cn(\Phi_f, i\}$ is the set of components of the ith granular level, $C(\Phi_f, i, j)$ is the jth component and $cn(\Phi_f, i)$ denotes the number of components at this level.

$R(\Phi_f, i) = \{r(\Phi_f, i, j),\ i = 1 \ldots ln(\Phi_f, i),\ j = 1 \ldots rn(\Phi_f, i)$ is the set of basic relationship types between components of the ith granular level.
$r(\Phi_f, i, j)$ designates the jth relationship type at the ith level, rni being the number of relationship types at this level.

For all couple of levels $(LV(\Phi_f, i), LV(\Phi_f, j),\ ,\ i = 1 \ldots ln(\Phi_f, i)\ ,\ j = 1 \ldots ln(\Phi_f, i) - 1,\ avec\ j > i$ corresponds :

$V(\Phi_f, i, j) = \{v(\Phi_f, i, j, k),\ i = 1 \ldots ln(\Phi_f, i),\ j = 1 \ldots ln(\Phi_f, i),\ k = 1 \ldots nv(\Phi_f, i, j)\}$
the set of types of basic relationships between components of the ith granular level and those of the jth one.
nv_{ij} designating the number of relationships types between the ith and the jth level, $v(i, j, k)$ is the kth type of relationship between the ith level and the jth granular level.

To each level $LV(\Phi_f, i)$ corresponds:
$AN(\Phi_f, i) = \{Atn(\Phi_f, i, j), i = 1 \ldots\ ln(\Phi_f, i), j = 1\ \ldots\ lan(\Phi_f, i)\}$ the set of attributes which will be appended on considered granular level, where,
$Atn(\Phi_f, i, j)$ is the jth attribute of the ith level, and

$lan(\Phi_f, i)$ is the number of attributes of the ith level.

To each $r(\Phi_f, i, j) \in R(\Phi_f, i)$ corresponds:
$AR(\Phi_f, i, j) = \{Atr(\Phi_f, i, j, k),\ i = 1 \ldots ln,\ j = 1 \ldots rn(\Phi_f, i),\ k = 1 \ldots nar(\Phi_f, i, j)\}$ the set of attributes of the jth type of relationship at the ith level, and $an(\Phi_f, i, j)$ is the number of these attributes.

To each $v(\Phi_f, i, j, k) \in V(\Phi_f, i, j)$ corresponds:
$AV_{ijk} = \{Atv(i, j, k, l),\ i = 1 \ldots ln(\Phi_f, i),\ j = 1 \ldots ln(\Phi_f, i),\ k = 1 \ldots vn(\Phi_f, i, j), l = 1 \ldots van(\Phi_f, i, j, k)\}$ the set of attributes of the kth type of relationship between the ith and the jth granular level, $Atv(\Phi_f, i, j, k, l)$ is the lth attribute, $van(\Phi_f, i, j, k)$ being the attributes number.

To each $SC(\Phi_f, i) \in LV(\Phi_f, i)$ corresponds $LT(\Phi_f, i) = \{TC(\Phi_f, i, j), j = 1 \ldots tn(\Phi_f, i)\}$, where $TC(\Phi_f, i, j)$ is the jth component type of the ith granular level, and $tn(\Phi_f, i)$ is the number of these distinct component types at the ith level where,
$(\forall\ C(\Phi_f, i, j) \in SC(\Phi_f, i)(\exists\ Type(C(\Phi_f, i, j)) = TC(\Phi_f, i, k) \in LT(\Phi_f, i)$.

To each $TC(i, j)$ corresponds the set of attributes
$ATC(\Phi_f, i, j) = \{Atc(\Phi_f, i, j, k), i = 1 \ldots ln,\ j = 1 \ldots tn_i,\ k = 1 \ldots an(\Phi_f, i, j)\}$, where $Atc(\Phi_f, i, j, k)$ is kth attribute of the component having the jth type of the ith level.

References

1. S. Ajila, H.Basson, and J.C. Derniame. A logic-based approach to impact analysis of objects change. In *Proceedings the Third International Conference on the Practical Application of Prolog*, pages 1–15, April 1995.
2. R. Al-Zoubi and A. Prakash. Program view generation and change analysis using attributed dependency graphs. *Journal of Software Maintenance: Research and Practice*, 4(7):239–262, July 1995.
3. Guillermo Arango and Rubén Prieto Daz. Domain Analysis and Software Systems Modelling. *IEEE Computer*, (1996), 1991.
4. D. Ash, J. Alderete, P.W. Oman, and B. Lowther. Using software maintainability models to track code health. In Hausi A. Müller and editors Mari Georges, editors, *Proceedings of the International Conference on Software Maintenance (ICSM '94) Victoria, B.C. USA*, pages 154-160, September 1994.
5. H. Basson. Contrôle de l'Évolution des Logiciels: Approche de Modélisation pour l'Analyse d'Impacts des Modifications. In *HDR Document*, pages 85–107, December 1997.
6. H. Gomaa and L. Kershberg. Domain modeling for software reuse and evolution. In IEEE Computer Society Press, editor, *CASE '95: Proceedings of the Seventh International Workshop on Computer-Aided Software Engineering*, pages 162–171, July 1995.

Metrics for Estimation

T. R. Judge, N. S. Mistry

Parallax Solutions Ltd, Coventry, UK.
http://www.parallax.co.uk/

Abstract

The Metrics in Object Oriented Developments (MOOD) project 21443 is focused on the introduction of metrics to aid Estimation on Object Oriented (OO) projects at Parallax. This is in line with meeting Parallax Solutions' requirement to improve its estimating processes and the requirements of a Process Improvement Experiment (PIE) as defined by the European Systems and Software Initiative (ESSI), under whose auspices the project is being run. For more information on this initiative see http://www.esi.es/ESSI/.

Improvement is to be achieved by:

- The sound application of project metrics to 4 baseline projects.
- Improvement of the whole metric gathering lifecycle.
- Development of metrics that aid the estimation process.

This will lead, at a minimum, to the business benefit of greater confidence in delivering OO developments on time to the satisfaction of the customer.

The project will initially involve a period of study, primarily to review information and material relating to estimation and metrics.

The PIE will then enter a cycle of metric development and constant process improvement. This process improvement will be applied on 4 selected OO projects within Parallax.

All projects undertaken by Parallax are developed in accordance with the Parallax Delivery Management System (DMS). The Estimation process defined in the DMS will be updated to include the recommendations of the project.

The PIE will be an iterative experiment constantly applying and reviewing the process and applying changes. Conclusions will then be drawn and reports quantifying the improvements will be completed and presented.

This paper presents the findings of the project so far in terms of metrics identified for estimation.

C. Hawkins et al. (eds.), *Software Quality Management VI*

1 Introduction

This paper concerns metrics for estimation on OO projects. As well as providing a description of the metrics that have been identified, the paper also presents a case study based on the MOOD project and an analysis and justification describing how and why these particular metrics have been selected.

The intention in capturing such metrics is to improve the estimating process on OO projects within Parallax Solutions Ltd.

The project has adopted the Goals, Questions, Metrics (GQM) paradigm defined by Basili and Weiss[1]. This paradigm provides a framework for the definition of an appropriate set of metrics based on project goals. However, it has been necessary to be somewhat pragmatic in terms of the metrics defined as it has been necessary to consider factors such as the project overhead involved in metric collection, and the capabilities of the metric software tools available.

This document draws on data gathered during the project activities listed below:

- Metric Audit - an audit of all Parallax projects has been conducted to determine the quality of metrics that are already available.
- Period of Study - The project has involved a period of study, to review information and material relating to estimation and metrics.
- Software Evaluation - The software evaluation has looked at a number of packages to evaluate their capabilities both in terms of metrics capture and storage and in terms of generating estimates based on metrics. Process Engineer (from LBMS http://www.lbms.com/) and KnowledgePlan (from Software Productivity Research http://www.spr.com/) have emerged as extremely powerful tools in this regard.

The MOOD project objectives are:

- The sound application of project metrics to 4 baseline projects including the introduction of a metric improvement process in place on those projects.
- Improvement of the whole metric gathering lifecycle including ongoing improvement of quality metrics.
- Development of metrics that aid the estimation process leading to a sound and accurate process of estimation.

It is expected that the project will have a direct impact on the quality and accuracy of Parallax's estimation process in two main ways:

- The application of quality metrics and measures on OO developments will enable progress to be monitored more accurately allowing us to build a

historical profile of projects we have successfully completed, the actions that made them work and those that did not.
- Applying further metrics and this historical information during the estimation process will allow estimates to be arrived at with more confidence and accuracy.

From these rather high level statements it has been necessary to develop a set of project Goals from which to derive the metrics that are required. These goals have been reviewed with the MOOD Business Champion (BC) to confirm that they are in line with the requirements of the Parallax business. The BC, in this case, is the Parallax Delivery Manager.

2 Goals, Estimating Techniques and Metrics for Estimation

The goals for the project in a GQM sense have been defined such that they are either directly related to the estimating process or affect the capture and accuracy of estimates. From these goals, questions have been raised. The answers to these questions suggest the metrics that should be captured.

For the process of determining the goals, an investigation has been conducted into the groups that are affected by estimates. Parallax mainly does work for outside clients who, when sending a tender for work, have a rough idea of the time-scales they would need the work completed by, this may be determined by other developments within their Company. Once the requirements are known, even at a high level, Parallax needs to give an idea of the estimated schedule, effort and size of the project. After detailed analysis Developers would be consulted on the estimated effort for the break-down of tasks. Once all the tasks have been estimated, it is the responsibility of the Project Manager to collate these estimates and form a project schedule highlighting major milestones and making sure the final end date is realistically achievable. After the Developers and Project Manager are happy with the plan, the final stage is to get agreement from the Customer that the schedule meets their expectations.

Five groups that are the beneficiaries of the estimation process have been identified as a result of the investigation. These groups are identified and discussed below, together with goals that would support these groups needs from the estimation process.

2.1 Beneficiaries of the Estimation Process and their Goals

1. Parallax - In general Parallax needs to analyse and improve its estimation process, this would better determine the success of projects and in turn improve profitability. In the long term Parallax can build on their reputation of

delivering projects on time and within budget so that Customers are more likely to come back with repeat business.

Goal	Rationale
Improve OO Software Estimation Process	• Improve the Parallax DMS process specifically for object-oriented projects • Improve accuracy of task estimation for object-oriented technology • Improve marketability of Parallax
Minimise Schedule	• Minimise non-development tasks • Plan company resource requirements • More time to develop new products and keep-up with latest technology

2. Project Team - If an accurate project estimate is available at the outset of a project then the project will run much more smoothly. The Project Manager will be free to spend his/her time on project problems that could not be predicted. The Customer will be happy to see that the project is progressing as planned and is therefore under control.

Goal	Rationale
Improve project estimation, schedule, effort and size	• Accurate OO project proposals • Ensure profitability • Maximise requirements stability • Minimise risks
Keep projects on track	• Avoid cost overruns • Avoid schedule overruns • Predict need for corrective action • Ensure conformance to standards • Give an accurate indicator of project progress

3. Developers - Realistic schedules mean the developers can spend time producing quality code, which is well designed, efficient and has been thoroughly tested. Once tasks are accurately recorded the next time a similar project is being estimated the historical task information can be used to more accurately predict the work that will be necessary.

Goal	Rationale
Improve individual estimating ability	• Feed into project estimates and tracking • True view of project progress • Early indicator of scheduled effort error/ unscheduled tasks
Improve skill levels to reduce developer time spent on tasks	• Highlight training need • Indicator skill shortage • Improve Company skill set

4. Project Manager - It is usually the task of the Project Manager to create and finalise the project plan. But there are many more responsibilities that sometimes get forgotten because all their time is spent with the plan. Selecting

the correct resources with the correct skills is very important to the project, the right skills can help meet tight timescales. A good start to a project is always very important, it is usually at the beginning that the important decisions are made, the software to be used, the methodology required, the level of Customer time needed, indeed more importantly a good rapport with the Customer at the beginning pays dividends in the end. It is important that the Project Manager is responsive to events as they occur and not pre-occupied with other tasks.

Goal	Rationale
Optimise resource usage	• Match resource skills, team structure to project requirements so time not wasted training • Get the right skill for the right tasks at the right time, keep within estimated cost
Minimise disruption by providing the ideal working environment	• Get the right team environment with a shared elevated vision or goal • Project rotation to keep Developer motivation high • Avoid disruption so estimates are not impacted

5. Customer - One way the Customer has an idea of how well the project is going is to compare progress against the schedule. If the scheduled effort has been inaccurate from the beginning the Customer's perception of the progress is going to be wrong. An accurate schedule would avoid this as well as gaining confidence. By making risks known early to the Customer, there will be 100% committed to the project ensuring all necessary resources and information is made available to the project team.

Goal	Rationale
Get 100% agreement of project schedule	• Justify estimates from past project experience • Better to be realistic at the beginning even if not meeting Customer's time-scale rather than delivering late • Improve Customer relationship
Minimise project risks by getting total commitment from necessary people and information	• Decision made by right people at the right time so there is no delay in progress

The goals identified above will remain valid throughout the project even if new processes or metrics are introduced. Following the MOOD project and depending on the success of it, more goals can be added for other areas of the business and the cycle of estimating process improvement spread to all Parallax projects.

These goals cover a broad range of goals for estimating, for the purpose of the final metrics to capture it is necessary to concentrate on those that are of direct interest to MOOD. Another reason for being specific about the goals is to produce metrics that can be effectively analysed, without being overloaded with too much information. The following statement taken from [2] justifies this,

> NASA's Software Engineering Laboratory (SEL) has had an active measurement program for almost 20 years. One of the lessons SEL has learned is to spend more effort on analysis and less on data collection. In the early years of the program SEL spent about twice as much on data collection as on analysis. Since then, data collection effort has dropped by half, and SEL now spends about three times as much on analysis and packaging as on collection.

Therefore the fewer metrics that are collected the more time can be spent on the analysis and feedback of the findings. A recommendation by Edward Berard[3] is,

> A single software engineering metric in isolation is seldom useful. However, for a particular process, product, or person, 3-5 well-chosen metrics seem to be a practical upper limit.

If a goal is seen as a particular process, product or person then for each goal a limit of 5 metrics is adequate. He also highlights the fact that the most useful metrics may not be known ahead of time, therefore,

> When we first begin to study some aspects of software engineering, or a specific software project, we will probably have to use a large (e.g. 20 to 30, or more) number of different metrics. Later, analysis should point out the most useful metrics.

The second cycle of metrics capture, should highlight the most useful metrics. The goals however should still be valid for both cycles of metric capture, as the metrics for each goal are improved, in theory only the most useful metrics will continue to be captured. Therefore more goals can be added which will not off balance the time spent on data collection compared to analysis and feedback.

2.2 Selected Goals

Following is a list of selected project goals from those identified for each group above. In order to formalise the process a structure suggested by the CEMP project, ESSI project 10358 run by Helmut Woda has been used.

The template for GQM goals is:

Purpose of Study
Analyse some (objects: processes, products, etc.)
For the purpose of (why: understanding, managing, motivating, engineering, controlling, certifying, etc)
Perspective of study:
With respect to (quality focus: cost, correctness, defects, changes, reliability, user friendliness, etc.)
From the viewpoint of (who: user, customer, manager, developer, corporation, etc.)
Context of study
In the following context (environment: problem, people, resources, processes, etc.)

This formal approach was necessary to provide a clear and consistent description of the goals selected.
Six goals have been chosen, these are presented below:

1. Improve OO Software Estimation Process	2. Improve Project Estimation, Schedule, Effort and Size	3. Keep Projects on Track
Analyse the OO Software Estimation Process **For the purpose of** improving it **With respect to** the estimating techniques **From the viewpoint(s) of** the Project Manager, Delivery Manager **In the following context:** Parallax	**Analyse** the Project Schedule **For the purpose of** accurately specifying **With respect to** the tasks and sub-tasks **From the viewpoint(s) of** Project Manager, Developer **In the following context:** current project	**Analyse** the Project **For the purpose of** keeping on track **With respect to** re-estimating **From the viewpoint(s) of** Project Manager, **In the following context:** Parallax

4. Improve Individual Estimating Ability	5. Minimise Disruption by Providing the Ideal Working Environment	6. Get 100% Agreement of Project Schedule
Analyse the tasks **For the purpose of** accurately estimating **With respect to** their effort **From the viewpoint(s) of** the Developer **In the following context:** current project	**Analyse** the Project Management **For the purpose of** better managing **With respect to** the project needs **From the viewpoint(s) of** the Project Manager, Delivery Manager **In the following context:** current project	**Analyse** the Customer **For the purpose of** getting support **With respect to** the project estimates **From the viewpoint(s) of** the Project Manager, Delivery Manager **In the following context:** current project

Unfortunately there is not space here to go into detail about the mapping of all of these goals into questions and metrics. However, the goals above have been used to define a set of questions that must be answered to meet the involved parties' goals. In turn these questions have been used to select metrics which address the goals. Section 2.4 presents the metrics which have been identified.

2.3 Estimation Techniques

The first goal shown above is to analyse the OO software estimation process for the purpose of improving it with respect to the estimating techniques. In order to specify the metrics required for estimation it is clearly first necessary to understand what data is necessary to drive the available estimating techniques.

A report by the Software Productivity Research Centre entitled 'The Impact of Software Cost Estimating on Projects that fail or succeed' states that one of 'the attributes most strongly associated with successful software projects included the usage of automated software cost estimating tools'. The MOOD project has investigated two automated estimation tools and the data required to drive them. These are Process Engineer (from LBMS http://www.lbms.com/) and KnowledgePlan (from Software Productivity Research http://www.spr.com/). Both are very powerful tools, whose scope is too broad to go into here. However, it is necessary to understand something of the estimating techniques that have been considered. These are explored below with respect to LBMS. KnowledgePlan relies on a different approach. It characterises the project that you wish to produce an estimate for and compares it with its database of approximately 6,700 (and growing) projects to obtain an estimate.

It appears from the literature that there are, in fact, very few estimating techniques targetted at OO developments. It should be understood that the MOOD project is not a research project. It is a Process Improvement Experiment (PIE). This is important to understand as the project is not attempting to define a new OO project estimating technique. It is, however, attempting to apply industry best practice and improve the estimating process on OO projects. Once the project has completed there may be some scope for correlating the OO metrics captured with project duration, but this is not the primary aim of the project.

The following methods are supported by LBMS Process Engineer and have been considered in the definition of the project metrics. There is an Object Based estimating technique, but this is not particularly sophisticated.

2.3.1 RiskContingency

In order to build an effective strategy for a development project, it is necessary to identify and understand all the factors which have the potential to influence the successful completion of each of the stages. This method is a structured attempt to identify and grade areas of uncertainty throughout the life of a development project. If potential threats can be anticipated, attention can be given to finding a remedy at the right time.

The purpose of the RiskContingency method is to identify and quantify areas of uncertainty within a project. This allows the project manager to recognise the extent to which the project is at risk and to take appropriate action to limit its exposure to these risks. It is important to view the numbers obtained from using this method as relative to one another. This will indicate whether the risk factor is low, moderate, high or extreme.

The RiskContingency method should be applied as early as possible in the project. If this is done in the early stages, the results will, of course, be more accurate and useful to the project manager. Likely actions will result such as increasing estimates, adding activities, planning risk areas in more detail, extending the time frame and assigning more experienced resources to certain areas. The project manager can then feel confident that the project can be completed within its allocated time scales and can demonstrate how he has arrived at his conclusions. He will have documented proof which means that he is fully accountable.

2.3.2 QuickEst

This is a top down effort estimating method for an entire project. Typical scenarios are described to help the project manager decide on the size of the project. By considering whether the project is considered to be either small, medium or large and historic data the QuickEst metric provides a value in effort days for the project. These values can be modified depending on what the organisation considers to be a small, medium or large project. This metric provides a ball park figure only and is useful in so far as it gives a general idea of how much effort may be involved in a certain project. It enables project managers to see if a project is viable from the start, thus saving him valuable time.

Further estimation should always be carried out following the QuickEst so that task levels can be validated and tasks apportioned. LBMS recommend that this is done using a more accurate bottom-up procedure.

2.3.3 Weighted Average (WAVE)

This method provides the user with an entry level procedure that can be used to predict project characteristics, such as work effort. The result gives a weighting of four times the most likely scenario whilst taking into account a measure of deviation representing possible extremes.

WAVE can be used at the beginning of a project and at any time during the project's life in order to make necessary modifications. It can be used either to calculate high level activities and then apportion values down to lower level activities or to calculate lower level activities and then aggregate higher level tasks.

Again, by using this method, the project manager is demonstrating his accountability in the planning process. WAVE gives an easy introduction to formalised estimating and therefore can be used within an organisation who are not accustomed to working within a quality environment. It requires minimum learning time and can be adapted to suit any project.

2.3.4 Effort plus Risk

To make further accurate assessments, contingency needs to be built into the estimating process. This will then show the additional amount of effort which needs to be added to allow for any expected inaccuracy of the estimate as a whole. The contingency level needs to be matched to the magnitude of risk in the estimate as a whole. Process Engineer provides a default percentage that calculates a contingency percentage. This can be adapted by users to provide a more customised view depending on the organisation.

Contingency is an important part of the overall process as, during estimating, allowances need to be made for every scenario. It contributes positively to the overall success of the project by enabling the project manager to show that he has thought of and allowed for any anticipated problems that may arise.

2.3.5 Function Points : IFPUG & Function Points : MK11

Using counting processes that were developed by Alan Allbrecht of IBM in the late 1970s and Charles R. Symons top-down estimating models can be used depending on the complexity and size of the system to be built. Calculations are made from the functional and technical characteristics of the system and the development environment. For this, an organisation needs to have some historical data to input plus the services of an experienced Function Points practitioner. The resulting figures which are obtained from these methods are more accurate due to their Top-Down nature.

2.3.6 Object Based Estimating

This method provides estimates of effort for low level activities. These values can then be aggregated for higher levels. It can be viewed as a useful cross-check against top-down estimation. Due to the level of detail required, this kind of estimation is recommended for tasks within a single stage of the project, as opposed to the project as a whole.

As estimation is to be carried out at the object level, Process Engineer provides an Estimating Variable List which consists of typical objects that are required for application development – data entities, data elements, data flows, transactions, programs, test paths and team members etc. Each of these variables can be estimated separately and the results added to the overall estimation picture gained from the other methods.

2.4 Metrics

Table 1. lists a superset of the metrics that will be applied in the course of the MOOD project. This is a long list, and as mentioned earlier NASA have observed that it is better to capture a small set of metrics and spend time analysing the data carefully than to capture vast amounts of information with little analysis. This is understood, but MOOD is a Process Improvement Experiment.

Over the course of the project this list will be refined to provide the information that Project Managers really need when estimates need to be produced. Due to space the table only gives a description of the metric and an indication of where it would be useful, either when a particular estimating technique is to be applied or at a particular stage in the project, or as parameter input to a tool such as KnowledgePlan. Developers also maintain their own spreadsheet of tasks that they have been asked to estimate, so that they can keep track of their own estimates and actuals to improve their estimating skills, this is referred to as the Personal Spreadsheet in the table.

Metric Description	**Where Applicable**
Effort (person hours) per project size	QuickEst
Effort (person hours) per function point – the Productivity rate	Function Points
Initial estimate versus actual effort (person hours) for each DMS Component	Consensus
Initial estimate versus actual project schedule for each task, object, DMS component	Consensus
Initial estimate versus actual size of the software (new and reused)	KnowledgePlan, Function Points
Initial estimate of staff required versus actual staff levels (for each project, DMS component)	Resource Assignment
Total overtime hours	QuickEst
Software schedule equation factor	QuickEst
Maximum compressed schedule effort	QuickEst
Person hours spent on rework	Risk
Testing tools used by Developer	Expertise Modifier/KnowledgePlan
User experience of software development lifecycle	Expertise/KnowledgePlan
User experience of business area	Expertise/KnowledgePlan
Level of User involvement during Requirements process	Expertise/KnowledgePlan
Number of Users in acceptance testing	Expertise/KnowledgePlan
Use of debugging tool	KnowledgePlan
Time taken to acquire tool ready for use	Risk/KnowledgePlan
Familairity of Tool by Project members	Expertise/KnowledgePlan
Number of platforms (Client/Server)	KnowledgePlan
Development module type requiring rework	Revise Estimates/Risk
Complexity of coordination between platforms	KnowledgePlan
Level of security requirements	KnowledgePlan
Number of performance targets	KnowledgePlan
Number of levels of Management hiearchy	KnowledgePlan
Level of team morale	Risk/KnowledgePlan
Type of office environment	LBMS - Non effective human factor/ KnowledgePlan
Noise and Interruption level in Office	LBMS - Non-effective human factor/ KnowledgePlan
Number of development locations	KnowledgePlan
Level of detail of QA process	KnowledgePlan
Actual effort for each non-development task	LBMS - Non-effective human factor
Time taken to review test plans	QuickEst, KnowledgePlan
Number of Test plans	KnowledgePlan

Level of experience of documentors	KnowledgePlan
Type of documentation e.g Comments, Application manual	KnowledgePlan
Actual hours on project related work vs non-productive work per person	LBMS - Non-effective human factor
Percentage of identified risk vs actual risk	Revise Estimates/Risk
Percentage of "Total cost" to "Cost of construction"	Effort Weighting for high level tasks
Percentage of construction costs for each of the other development costs	Effort Weighting for high level tasks
Defects by classification, severity, subsystem, phase of testing	Defects data
Number of resource by project size	Resource Assignment
Software product complexity, project type category	KnowledgePlan
Actual vs estimated project schedule per project type	Consensus
Actual effort (person hours) for each object	Object-based Estimating
Person hours spent on investigation, background reading, training	LBMS - effort weighting
Effort (person hours) required to investigate re-use components	Effort Weighting per Task
Level of testing on components identified for reuse	Testing data
Level of complexity of components identified for reuse	Function Points
Level of expertise of developers that wrote reuse components identified for reuse	Risk
Person hours spent on tasks related to particular technology	Consensus/KnowledgePlan
Range of task list totals	LBMS - QuickEst
Initial task list totals vs actual task list totals	LBMS - QuickEst
Size of task (days effort) with most accurate estimates	Object-based Estimating
Average error in the function point cost	Function Points
Average error in the testing point cost	Function Points
Level of coupling between object classes or dependency of functions	KnowledgePlan
Time taken to test object classes and related functions	Testing data
Number of change requests for each task, object and DMS component	Revise Estimates
Number of lines of code per month	Revise Estimates
Number of functions or objects per month	Revise Estimates
Number of tasks per DMS component	QuickEst
Number of tasks per resource	QuickEst
Number of reuse components vs new components	KnowledgePlan
Number of hours per page of documentation	KnowledgePlan
Percentage of tasks complete	Revise Estimates
Earned value for each task, DMS component	Financial data
Percentage of budget spent to date	Financial data
Number of risks per DMS component	Risk
Time period of each risk	Risk
Contingency vs actual impact of risk	Risk
Actual schedule vs initial estimate per developer task	Personal Spreadsheet/Revise Estimates
Number of open defects vs total defects	Defects data
Person hours per defect, severity, subsystem	Defects data
Reason for re-scheduling	Revise Estimates
Number of man-months to complete	QuickEst
Number of months in schedule	QuickEst
Initial effort estimate vs actual effort per task	Personal Spreadsheet/Revise Estimates

Estimated object interfaces vs actual object interfaces	KnowledgePlan
Number of change requests per developer	Personal Spreadsheet/Revise Estimates
Number of code reviews	Personal Spreadsheet/Revise Estimates
Accuracy of estimates vs role description	Risk
Accuracy of Developers task estimates per DMS component	Revise Estimates
Estimated tasks vs actual tasks per Developer	Bottom-Up
Initial bug fix estimates vs actual effort (person hours) per developer	Defects data
Number of days slipped per resource change	Staffing data
Number of days specialist skills used per project type, per project size	Resource Assignment
Percentage of project complete vs DMS phase resource left project	Staffing data
Skill level of developers per DMS phase	LBMS - Resource Assignment Expertise Modifier
Date of last developer skill level update	Expertise Modifier
Developer skill level per area of rework	Risk
Number of key hygeine factors satisfied per DMS component	Risk
Number of motivational factors achieved per DMS component	Risk
Number of key team cohesion factors per DMS component	Risk
Number of days spent on familiarisation	Effort Weighting per Task
Customer Business Area	KnowledgePlan
Time taken by customer in accepting the estimates	Risk/Consensus
Increase/decrease in schedule by Customer	Risk/Consensus
Increase/decrease in function points to meet Customer's deadline	Function Points
Level of customer satisfaction vs rate of project progress	Revise Estimates
Project Nature	KnowledgePlan
Project Scope	KnowledgePlan
Project Topology	KnowledgePlan
Project Type	KnowledgePlan
Algorithmic complexity of problem	KnowledgePlan
Code Efficiency	Function Points, KnowledgePlan
Data Structures	Function Points, KnowledgePlan
Sizing Methods	KnowledgePlan
Project Size	QuickEst, KnowledgePlan
Project Manager Name	KnowledgePlan
PM level of use of Estimating tools	KnowledgePlan
Formal Method used by Analyst	KnowledgePlan
Programming Language used by Developer	Expertise Modifier/KnowledgePlan
Development or target platform used by Developers	Expertise Modifier/KnowledgePlan
Tools/Methods used by Developers	Expertise Modifier/KnowledgePlan
Developer experience of doing code reviews	KnowledgePlan

Table 1. Metrics for Estimation

3 Conclusion

This paper has presented some of the findings of the MOOD project to date. The paper has covered:

- A set of goals defined to meet the needs of Parallax and those within the organisation who have a stake in accurate estimation.

- Estimating techniques and automated tool support.

- The metrics required to drive the improvement of the estimating process and support the available techniques of estimation and risk analysis.

4 References

1. Basili VR, Weiss DM, A Methodology for Collecting Valid Software Engineering Data, IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, 1984, Vol.10, No.6, pp.728-738.

2. McConnell S, Rapid Development, ISBN 1-55615-900-5.

3. Berard E, Essays on Object-Oriented Software Engineering, Volume {I}, Prentice Hall, 1993, ISBN 0-13-288895-5.

AUTHOR INDEX

Allen P. 89
Banga K.S. 230
Basson H 260
Bennetts P.D.C. 38
Bicego A. 26
Bradley M. 209
Brinkworth J. 3
Burgess W.P. 204
Cater-Steel A.P. 187
Chan K.C.C. 161
Cogan B.I. 173
Coleman G. 241
Dawson R.J. 209
Edwards H.M. 108
Fitzgerald E.P. 187
Georgiadou E. 135
Hardy C.J. 108
Heather M.A. 218
Hericko M. 147
Horvat R.V. 147
Hung K.S.Y. 121
Judge T.R. 270
Juric M.B. 147
Kaikkonen T. 15
Khurana M. 26
Kuvaja P. 26
Lau J. 53
Llewhellyn J. 3
Lykouropoulos N. 106
Macmillan R.H. 199
Maniatis A. 106
Matveeva T.O. 173
Milankovic-Atkinson M. 135
Mills S. 38
Mistry N.S. 270
Oivo M. 15
Puputti J. 15
Rose A. 121
Rossiter B.N. 218
Rozman I. 81,147
Simons A.J.H. 121
Smart P. 70
Thompson J.B. 108
Verbruggen R. 241
Vierimaa M. 15
Welzer T. 81
Wilson D.N. 53
Wood-Harper T. 38
Yan C.Y. 161