A Quality Software Process for Rapid Application Development

By

Gerard Coleman

School of Computer Applications,
Dublin City University,
Glasnevin,
Dublin 9.

M. Sc., September 1997

A Quality Software Process for Rapid Application Development

A Dissertation Presented in Fulfilment
of the Requirement for the M.Sc. Degree

September 1997

Gerard Coleman

School of Computer Applications
Dublin City University

Supervisor: Mr. Renaat Verbruggen

Declaration

I hereby certify that this material, which I now submit for assessment on the programme of study leading to the award of M.Sc. is entirely my own work and has not been taken from the work of others save and to the extent that such work has been cited and acknowledged within the text of my work.

Signed: Gerard Coleman **ID No.:** 94971480

Candidate

Date: 29th September 1997

Acknowledgements

I would like to thank Renaat for his assistance, insight and guidance during the research.

I would also like to thank my wife, Fiona, for her encouragement and understanding throughout the compilation of this work.

Finally, I would like to thank Patrick O'Beirne, Shay Curtin and Gerard McCloskey who gave so generously of their time to review this work.

Abstract

Having a defined and documented standardised software process, together with the appropriate techniques and tools to measure its effectiveness, offers the potential to software producers to improve the quality of their output. Many firms have yet to define their own software process. Yet 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 operating using 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.

DSDM has been designed as a framework for Rapid Application Development (RAD) and provides a documented approach for organisations to follow when undertaking RAD projects. Through the mechanisms outlined by DSDM developers become empowered and time-to-market for software can be substantially reduced.

The PSP allows individual software engineers to assess, measure and improve their performance. By improving the skills of individual developers, quality can be engineered into RAD projects at all life-cycle stages.

Combining PSP and DSDM, therefore, enables the production of high-quality software and at the same time allows reductions in development time to be achieved.

Table of Contents

Chapter 1 - The Software Process 1

1.0 Introduction 1

1.1 What is the Software Process? 2

1.2 Background 3

1.3 The Software Process - Maturity and Assessment 6

1.3.1 Process Maturity 6

1.3.2 A Maturity Framework and the Capability Maturity Model ... 7

1.3.3 'Bootstrap' 9

1.3.4 SPICE 10

1.4 Dynamic Systems Development Method (DSDM) 10

1.5 Personal Software Process (PSP) 11

1.6 Summary 11

Chapter 2 – Rapid Application Development (RAD) and the Dynamic Systems Development Method (DSDM) 12

2.0 Introduction 12

2.1 Rapid Applications Development (RAD) 12

2.2 RAD – A Worthwhile Investment? 12

2.3 Dynamic Systems Development Method – A RAD Standard 19

2.4 DSDM – What the Method Contains 20

2.5 Principles of DSDM 20

2.5.1 Applications Suited to the DSDM Approach 21

2.5.2 Applications Unsuited to the DSDM Approach 21

2.5.3 DSDM Implementation - Critical Success Factors 21

2.6 The DSDM Life-Cycle 22

2.6.1 Prototyping within DSDM 24

2.7 Quality Issues in DSDM 25

2.7.1 Quality Control 25

2.7.2 DSDM and the CMM 25

2.8 Introducing DSDM into an Organisation 25

2.9 DSDM - A Way Forward for RAD? 26

2.9.1 Other Factors which May Influence Development 29

2.10 DSDM - The Benefits 32

2.11 Summary 34

Chapter 3 – Process Improvement using the Personal Software Process (PSP) 35

3.0 Introduction 35

3.1 Personal Software Process (PSP) 35

3.2 PSP Improvement Phases 36

3.3 The Baseline Personal Process 36

3.3.1 PSP0.1 37

3.4 PSP1 38

3.4.1 Proxy-based Estimating 38

3.4.2 Resource and Schedule Estimating 39

3.4.3 Measurements in the PSP 39

3.5 PSP2 39

3.5.1 Design and Code Reviews 39

3.5.2 Software Quality Management 40

3.5.2.1 Product Quality 40

3.5.2.2 Process Quality 40

3.6 PSP3 41

3.6.1 Scaling Up the Personal Software Process 41

3.6.2 The PSP3 Approach 41

3.7 Using the Personal Software Process 42

3.8 Personal Software Process - An Analysis 43

3.9 Summary 49

Chapter 4 – Assessment of the Software Process within a Small Company 52

4.0 Introduction 52

4.1 Software Process Questionnaire 52

4.2 Questionnaire Findings 53

4.2.1 Project Review and Sign-Off 55

4.2.2 Configuration Management 56

4.2.3 Software Estimation 56

4.2.4 Metrics 56

4.2.5 Education/Training 57

4.2.6 Project Management 57

4.2.7 Software Quality Assurance 58

4.2.8 Requirements Specification 58

4.2.9 System Design 59

4.2.10 Implementation 59

4.2.11 Testing 60

4.2.12 Operations/Maintenance 60

4.3 Analysis of Findings 60

4.4 Summary 62

Chapter 5 – Evaluation of the Development Environment within a Small Software Company 63

5.0 Introduction 63

5.1 Process Assessment - Environment Evaluation 63

5.2 Development Strengths and Weaknesses within the Company 66

5.2.1 Development Strengths 66

5.2.2 Development Weaknesses 66

5.3 Comments on the Development Process 69

5.4 Starting Point for an Improved Process 69

5.4.1 The Requirements Document 71

5.4.2 The Test Plan 72

5.4.3 Using Fault Reporting 72

5.4.4 Change Request Form 73

5.5 Summary 74

Chapter 6 – A Software Process for RAD 76

6.0 Introduction 76

6.1 Implementing a Software Process for RAD 76

6.2 Using DSDM within the Company 78

6.3 How Will using DSDM Benefit the Company? 80

6.4 Using PSP within the Company 82

6.5 A New Development Environment 83

6.5.1 A Quality Software Process for RAD – Combining DSDM and PSP3 84

6.5.2 A Quality Software Process for RAD - Using Proxies 86

6.5.3 A Quality Software Process for RAD - Testing 88

6.5.4 A Quality Software Process for RAD – A Quality Plan 89

6.5.5 A Quality Software Process for RAD – Metrics 91

6.5.5.1 Defect Metrics 94

6.5.5.2 Productivity Metrics 96

6.5.5.3 Size Metrics 98

6.5.5.4 Time/Schedule Metrics 99

6.5.5.5 Extended Measures 102

6.5.5.6 Cost of Quality (COQ) 104

6.5.6 Maintenance 105

6.5.7 Software Reuse 107

6.6 Summary 109

Chapter 7 - Independent Review 111

7.0 Introduction 111

7.1 Review Panel 111

7.2 Independent Review Comments 111

7.3 Summary 118

Chapter 8 - Conclusions 119

8.0 Introduction 119

8.1 Detailed Analysis 119

8.2 Recommendations 121

8.2.1 Data Recording Assistance 121

8.2.2 Inspection/Review Assistance 122

8.2.3 DSDM Software Process Improvement Measures 122

8.3 Future Research 122

9. References 125

Appendix A 130

Appendix B 131

Appendix C 132

Appendix D 135

List of Figures

1.0 Waterfall Model 4

1.1 Spiral Process Model 5

1.2 Process Maturity Framework 8

2.0 DSDM Life-Cycle 23

2.1 DSDM Prototyping Cycle 24

3.0 PSP Evolution 36

3.1 PSP3 Process 42

4.0 Company Process Maturity Chart 55

5.0 Company Development Paradigm 65

List of Tables

2.0 Advantages and Disadvantages of Prototyping 18

2.1 Advantages and Disadvantages of RAD 19

2.2 DSDM - Critical Success Factors 22

2.3 Summary of Inspection Data 27

3.0 Object Category Size in LOC 38

5.0 Company Strengths and Weaknesses 66

5.1 Company Weaknesses - Standardisation and Procedures 67

6.0 Factors which favour DSDM Usage 79

6.1 Factors which favour PSP Usage 83

6.2 Defect Type Standard 93

6.4 List of Metrics 94

Chapter 1 - The Software Process

1.0 Introduction

This study examines the different approaches which can be taken to develop software with a view to proposing a methodology for use in small software companies. Many companies possess a backlog of software waiting to be developed. Much of this backlog exists because companies must devote large resources to maintaining existing software. This can occur, either through enhancement of existing systems or, through having to fix systems of poor quality.

These problems are particularly acute in small companies.

To try and reduce the backlog and address this 'software crisis' companies are using a number of approaches. These include improving the quality of developed software so that less time and resources are spent fixing defects and using Rapid Application Development (RAD) techniques to speed up the development process [MART91].

The objective of this study is examine this 'software crisis' as it exists in small software companies and to propose a software process framework suitable for such companies who wish to get quality software products onto the market as quickly as possible. The research involved the study of software processes and methods suitable for use in small companies. Also a small software development company was examined to determine the methods and processes it used and subsequently how any suitable software process paradigms might be applied within such an environment.

The study concludes by proposing the use of a combination of the Dynamic Systems Development Method (DSDM) [DSDM95], a life-cycle for use in RAD environments, and the Personal Software Process (PSP) [HUMP95] which is targeted at improving the performance of individual software developers.

1.1 What is the Software Process?

Many companies, particularly small and medium-sized enterprises (SMEs), develop software in an unstructured and undefined way. The set of approaches and mechanisms which organisations use to develop software is known as The Software Process.

Every organisation has its own particular software process. Some are based on traditional development models such as the Waterfall Model [ROYC70] whilst others have evolved from historical company practices.

In order to improve its software capability each organisation must have a defined and documented software process. However, many different process models exist and have applicability in different environments. Some models, such as the Capability Maturity Model (CMM), assess the 'maturity' of software development organisations [HUMP88]. The CMM basically indicates that the more mature a software development organisation, the more capable it is of developing high-quality software.

Unfortunately, many companies operate without a defined process, with each software system being developed independently without regard for previous or concurrent developments.

As such, the quality of the finished product depends primarily on the assiduity of the development personnel, their design and programming skills and commitment to comprehensive testing.

Also, any methods, or tools, to support the process are used in an unstructured and haphazard fashion. Further, it is unlikely that any metrics will be collected.

Boehm defined software engineering as:

'The application of science and mathematics by which the capability of computer equipment is made useful to man via computer programs, procedures and associated documentation' [BOEH76].

A process, Humphrey states, is a series of tasks, that when properly executed, achieves the desired result [HUMP88].

Ultimately, the manufacture of software can be reduced to a list of tasks that must be carried out in a certain sequence. However, if the desired result is software of measurable quality then it is necessary to adopt an engineering-style approach to the creation of that product.

1.2 Background

It has taken a long time for the idea that software should be developed within a mature, managed process, to become established.

In 1987, Brooks claimed that, at that time, there did not exist a single development in either technology or management technique that promised a significant improvement in software engineering but, notwithstanding that, a coherent and consistent attempt to exploit and incorporate new technologies should produce that improvement [BROO87]. Prior to the emphasis on the software process and software engineering approaches, designing an appropriate model for software development was the major task.

In 1956, Benington proposed the stagewise model which suggested a series of phases along which development would proceed [BENI56].

Royce expanded on this to produce the Waterfall model [ROYC70]. The Waterfall model became the standard approach to software development for many subsequent years and is still in wide use today.

The layout of the Waterfall model is shown in **Figure 1.0.**

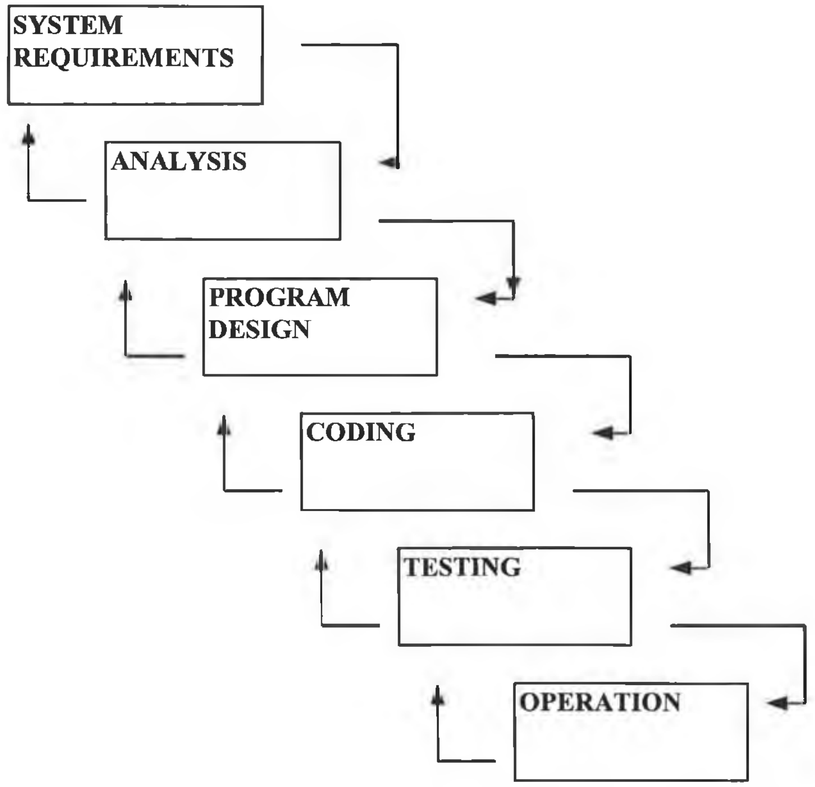

Figure 1.0 - The Waterfall Model

What the model shows is essentially a series of progressions through discrete stages, from system inception through definition of requirements, analysis, design and coding, testing and then final delivery of systems.

On the whole the model has proved a successful management tool and has laid the foundation for structured software development, however, its rigid hierarchy and inflexibility has led to the creation of alternatives.

In attempting to address the deficiencies contained within the Waterfall model, Balzer et al. proposed the Transform model [BALZ83]. The Transform model is based on the production of a formal specification of a software requirement and the subsequent conversion of that specification into a working program matching the specification.

The model also incorporates the facility to amend the specification based on operational experience. The advantage here is that, as the code is regenerated, it always matches the specification.

However, this technique may only be used at present in limited application areas and unplanned evolutionary paths may be difficult to incorporate.

In 1986, Barry Boehm developed what he termed a 'Spiral' model (**Figure 1.1**) for software development, which though essentially based on the Waterfall model, attempted to incorporate the best features of all the preceding models [BOEH88].

The key difference with the Spiral model and its predecessors is the fact that it is risk-driven, rather than document- or code-driven. The Spiral model moves from the centre out and works as follows :

Each stage begins with determining the objectives for that stage, the alternative approaches which can be taken and the evident constraints. The alternatives are then evaluated and the associated risks are identified and resolved.

Development and verification of the product then follows and finally planning of the next phase is carried out. Each cycle of the spiral terminated by a review.

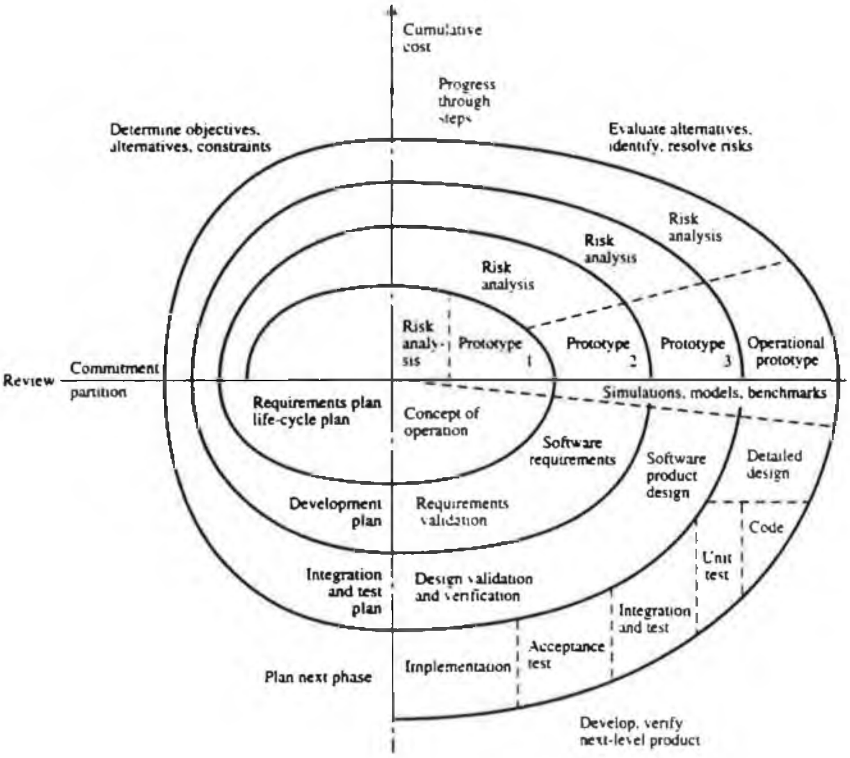

Figure 1.1 - The Spiral Process Model [BOEH88]

The exposed limitations of the Waterfall model led to the formulation of the Evolutionary Development model [McCR92]. This model is well placed to take advantage of fourth-generation languages as essentially an initial version of a software product is produced rapidly, evaluated, the amendments incorporated and a new version speedily created. However, the evolutionary development model possesses its own limitations in that its **code/test/fix** procedure can lead to a planning deficit.

Software development or life-cycle models can teach us a lot about the creation of software and the steps required. However, to ensure continued success across all development areas, these models need to be incorporated as part of an effective software process.

1.3 The Software Process - Maturity and Assessment

In the 1980s, when it became apparent that deficiencies in managing development and maintenance were hindering the improvement of software productivity and quality, the move towards a software process began.

1.3.1 Process Maturity

Process maturity is intrinsically linked to the concept of quality management.
Thompson and McParland state that a mature organisation has a well-defined software development process and measures both the quality of the process and the products it creates [THOM93]. Gauging the maturity level of an organisation is achieved by using assessments to analyse the competence or capability of an organisation's development process.
Curtis and Paulk proceeded to differentiate the mature organisation from the immature one [CURT93]. Accordingly, they state that the mature organisation possesses the ability to meet its cost, quality and schedule targets.

However, the failures of the immature organisation are legion:

⇒ Processes are improvised during projects.

⇒ Unrealistic assumptions are made about project and phase completion dates.

⇒ Because of these unrealistic assumptions product functionality is often jettisoned in a desperate attempt to meet deadlines.

⇒ Success depends totally on the commitment and talent of the developers.

⇒ The final products often contain many errors, incomplete documentation and the delivery lacks rudimentary configuration management.

The mature organisation, by contrast, has a well-defined, carefully managed process in place and this process is communicated to all employees. Thus, employees have a clear understanding of their roles/responsibilities.

All developments are planned and the process ensures the plan is adhered to in all respects. Quantitative methods are used to judge the quality of the software product.

Finally, when new technology is to be introduced or developed, the process can be simply adjusted to cater for this.

1.3.2 A Maturity Framework and the Capability Maturity Model

Several bodies, most notably the Software Engineering Institute (SEI) have been in the vanguard of promoting assessments of organisations and the evaluation of organisations' software development capability.

The SEI developed a Maturity Framework which included two methods, software process assessment and software capability evaluation, plus a maturity questionnaire to appraise software process maturity [HUMP88].

The assessment helps in finding the maturity level of the organisation's process while the questionnaire examines in great detail all aspects of software development including methodologies and tools used.

Endeavouring to illustrate this improvement path for customers, the SEI categorised organisations under five headings denoting their level of maturity [**Figure 1.2**].

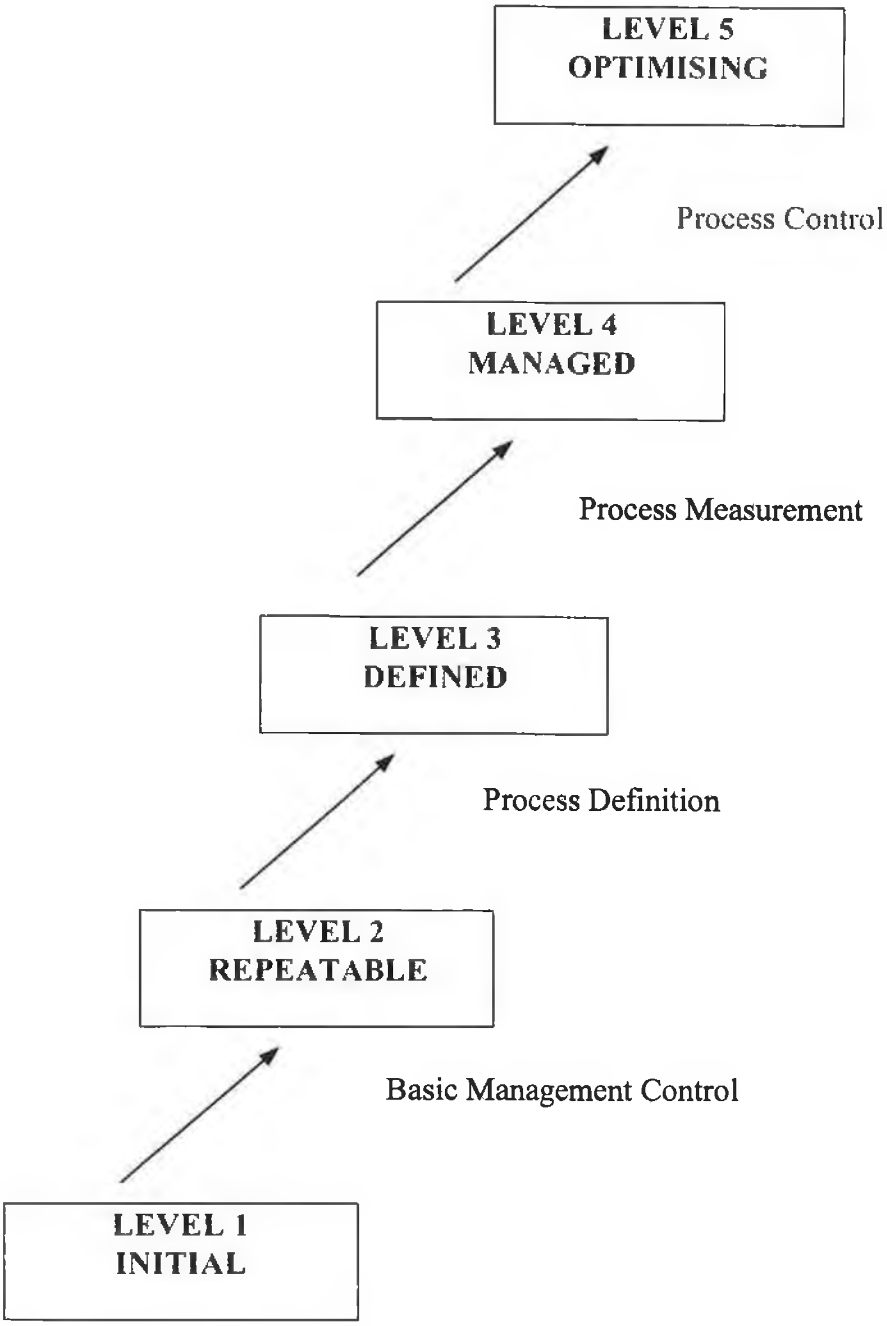

Figure 1.2 - Process Maturity Framework

At Level 1 ***'Initial'*** the development process is chaotic and unstructured.

From levels 2 through 5 organisations are attempting firstly to establish and then subsequently improve their development process.

At **Level 2** ***'Repeatable'*** , the overall objective is to ensure that successful techniques and approaches utilised on previous projects can be assimilated and used on current and indeed future developments.

Level 3 ***'Defined'*** - At this level companies now focus, rather than on specific projects or project management techniques but on process and ensuring its integration throughout the organisation. The emphasis now is on process definition and improvement and on equipping all software engineers and managers with the skills and tools required to execute their roles effectively.

Level 4 - ***'Managed'*** - Here, software measurement is introduced to assist in managing the quality of the process itself and the software product.

Level 5 - ***'Optimising'*** - Having achieved the previous four maturity levels, the organisation can now focus on continuous quality improvement.

After extensive work in this area the SEI evolved the Maturity Framework into the **Capability Maturity Model (CMM)** [PAUL93]. The **CMM** is based on the knowledge acquired from these studies and specifies recommended practices in the particular areas that have been shown to enhance software development and maintenance capability.

1.3.3 'Bootstrap'

It would be incorrect to conclude that the work being carried out by the SEI was the only development in this area. Work has recently been ongoing in Europe.

'Bootstrap' was a project completed as part of the European Strategic Programme for Research in Information Technology. Its objective was 'to develop a method for software process assessment, quantitative measurement and improvement' [HAAS94]. In doing so, 'Bootstrap' took the model developed by the SEI for process assessment and tailored it for use in the European Software arena. The key aspects of 'Bootstrap' are :

- Questionnaires are created to establish the organisation's/project's maturity level.
- Organisations are encouraged to create a software engineering process model before setting up a quality system.

- Unlike the SEI method which expects certain attributes to be evident at certain maturity levels and to rate companies accordingly, 'Bootstrap' rates the attributes irrespective of the level at which they occur.
- The maturity of the organisation is more important than the maturity of the technology or the methodology.

1.3.4 SPICE

The **SPICE (Software Process Improvement and Capability dEtermination)** standard was launched by the International Standards Group for Software Engineering in 1991 [DORL93].

Its objective is to develop an international standard on software process assessment.

The project includes features of the CMM and 'Bootstrap' among others.

The standard can be used :

1. To help organisations evaluate the capability of a software supplier.
2. By companies to assess and improve their own development and maintenance process.
3. By companies to determine their ability to implement new software projects.

1.4 Dynamic Systems Development Method (DSDM)

The Dynamic Systems Development Method (DSDM) was established to define a process for use with projects developed using Rapid Application Development (RAD) techniques. It is an attempt to provide a life-cycle and process in which RAD projects can be managed, controlled and tracked. It appears suitable for companies involved in developments where time-to-market is crucial or where the user interface is of prime importance. The method will be discussed in more detail later in this study.

1.5 Personal Software Process (PSP)

The Personal Software Process (PSP) is an attempt to scale down the best, large-scale, software practices for use by individual developers. It enables individual practitioners to define, manage, measure and subsequently improve their own software process.

It has particular use in small software departments or organisations.

The PSP will also be examined in more detail later in the study.

1.6 Summary

The objective of this study is to look at the process operating within small software organisations in Ireland and in one small company in particular.

Many companies are currently developing software in an ad-hoc manner within an unstructured environment. Efforts to improve companies' development capabilities include CMM, Bootstrap and SPICE.

This study is concerned with the introduction of quality practices into small software organisations. The sample company examined in the study is a small software producer. Because of its size, and the market sector in which it operates, all the documented software improvement approaches are not applicable.

The implementation of the most suitable methods and techniques, which can be beneficially introduced into the company, will be outlined when the background and operational characteristics of the company have been described.

Chapter 2 – Rapid Application Development (RAD) and the Dynamic Systems Development Method (DSDM)

2.0 Introduction

This chapter looks in detail at Rapid Application Development (RAD). It begins by describing what Rapid Application Development is and examines a number of studies which have been undertaken in this area. These studies highlight the advantages and disadvantages of RAD, the pitfalls which can be encountered and the potential benefits of using the approach.

This section then progresses to examine the Dynamic Systems Development Method (DSDM) which has been promoted as a life-cycle framework for RAD. The section concludes by discussing whether DSDM achieves its objective of being a suitable life-cycle approach for RAD and discusses its applicability in small companies.

2.1 Rapid Applications Development (RAD)

The term Rapid Application Development or RAD has generally been attributed to the consultant James Martin, since the publication of the eponymous book [MART91].

It is taken to relate to projects based around tight timescales, which use prototyping and combine high-level development tools and techniques.

2.2 RAD - A Worthwhile Investment?

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.

In his spiral model, Boehm, was one of the pioneering proponents of using prototyping in software development [BOEH88]. It was, he felt, one way of reducing project risk.

According to Luqi and Royce, prototyping has three main benefits :

1. By demonstrating the user interface, it improves communication between users and developers.
2. By improving communication it reduces any risks inherent in the project
3. It assists in validating specifications; supporting a common understanding and agreement between developers and users thereby making it more likely that the system will satisfy user requirements [LUQI91].

Gordon and Bieman, in their study, claim that in 80% of cases developers considered rapid prototyping a success [GORD95]. The study focused on three headings, **Product Attributes, Process Attributes** and the **Problems** encountered in RAD projects. On the **Product Attribute** side respondents indicated that RAD projects improved 'Ease of Use' and in a significant majority of cases matched 'User Needs' better. On the **Process Attribute** side, of those who recorded a change, the reduction in development effort was a notable factor. Also, end-user participation was greatly increased. However, some **problems** were reported in the different areas.

Evolutionary prototyping possesses inherent difficulties particularly with the danger of inefficient code being retained and therefore being a part of the final product.

Quality problems can also occur because of the speed of development and the rate of change of the prototype. To counteract this, good control procedures, standards and documentation are necessary. Furthermore, while user involvement is essential for success in RAD projects this too has to be controlled.

Another highlight from the study, showed that more experienced development staff were required for RAD projects as programming in this area regularly involves design decisions. Finally, Gordon and Bieman conclude that rapid prototyping can be a positive factor in software development and can be used in a variety of applications.

Other RAD supporters are also positive about its future. Reilly contends that RAD can benefit both users and developers [REIL95]. He believes RAD approaches will help ensure a business and user-oriented approach and thus ensure that all requirements are properly gathered; concurrent development activities will reduce system delivery times; and evolutionary prototyping extends prototyping into the analysis stage thus improving the prospects of building the 'right' system.

Carmel, takes issue with some of these claims [CARM95]. He believes that RAD approaches will not work because the following are not correctly addressed; Picking the Right Team, the Management and Customer Support of the project and the Methodologies used. On 'selecting the right team' he essentially concurs with Gordon and Bieman in that getting the right people is important for project success and that technical skills are paramount. Also, because of the extent of user interaction, good communication skills are also important. On 'management and customer support' he is referring to sound management understanding and control of the project. Also, almost in a mirror of that required of developers, customers have to be available but also able to make decisions quickly. On the third issue of Methodology, he asserts that RAD projects abandon rigorous methodologies and because of this software reuse becomes impossible later on.

Card, examines RAD in a business context concentrating on whether RAD helps in getting products onto the market more quickly [CARD95]. He believes that incremental development used in RAD approaches may reduce time-to-market in that the customer receives at least partial capability sooner. However, he also highlights the need for good project management of RAD projects.

Olsen, also concentrates on time-to-market in his examination of RAD [OLSE95]. He believes that the iterative development approaches, such as develop-test-repair, inherent in RAD, rather than the more traditional 'Waterfall' approach, can help reduce time-to-market.

Rafii and Perkins, look at RAD in terms of concurrent engineering and contrast this with conventional sequential development [RAFI95]. Concurrent engineering allows different stages of the development process to overlap, thus reducing development time. They caution, that even though project management is important to the success of all projects, it is crucial to the success of a project using concurrent engineering techniques. These findings are in agreement with those of Card, Carmel and Gordon and Bieman.

Henry and Faller, also looked at time-to-market as a reason for using RAD [HENR95]. Their belief was that time-to-market or cycle/development time could be reduced through software reuse as a RAD technique. Again, it is pointed out that project management, in conjunction with the appropriate development methods and tools, is essential to success. Their results show that through software reuse, cycle time can be substantially reduced, developer productivity can be increased and quality can be improved as a result of lower fault rates.

Linthicum, states that RAD promises the following advantages over traditional programming :

1. A reduced, more flexible development cycle and
2. Competent end-users can develop applications [LINT95].

He proposes that organisations should focus on the business objectives as this, he believes, is the way to good RAD usage. Another factor in good RAD development, he adds, is the use of reusable components. He does list some disadvantages to using RAD. Firstly, executing or porting RAD code can be time-consuming as the development tools are typically interpreters (e.g. Visual Basic) and will therefore execute much more slowly than compiled code. Secondly, you may also be locking yourself into a particular platform as, for example, two of the major RAD tools, Visual Basic and Delphi, only support Microsoft Windows. Thirdly, he refutes the claim made by RAD vendors that end-users can develop their own applications using RAD tools. With the exception of the simplest systems, he reckons developers must learn the underlying language. He concludes that RAD projects still need a good understanding of business requirements, a sound design and skilful programming.

Jacques, asserts that RAD developed systems are of a higher quality and are more flexible than those developed using traditional approaches [JACQ94]. Applications development is simpler, he believes, and there are benefits from users being involved from the beginning. Apart from the promise of increased system acceptance, this also allows users to develop a training plan for system usage.

He also suggests that an advantage of RAD is that the applications can be used as templates for future development efforts. He does warn, however, that to achieve success, staff must be trained in RAD concepts.

Simon, contends that, using traditional methods of systems development, problems are not discovered in the early stages of the life-cycle [SIMO95]. He counsels that RAD, in order to be fully successful, needs the development of comprehensive requirements, product specification details and data modelling. Also, he states, there may be some performance trade-offs in using RAD tools.

Hanna, in her study canvassed industry practitioners for their views on RAD [HANN95]. One respondent claimed that companies are turning to RAD because they can no longer afford the long requirements definition phase associated with the 'Waterfall' model. Also, the tools are currently available to allow developers to progress without having to know all the details of the application under development.

Furthermore, it is stated that projects, which last longer than six to nine months, risk a change in system functionality or loss of interest by the potential user. Another respondent suggested that RAD is not always the best way to develop software. If developers know a great deal about the systems to be developed the waterfall model should be the chosen approach. Conversely, though, if the requirements are vague and you need to clarify the user interface, then RAD is a very productive way to proceed.

One respondent declared that a developer's ability to work with the customer is crucial. During this process, the business rules of the system can be determined and the processing details can subsequently evolve.

A focused project study in the RAD field is Kerr and Hunter [KERR94]. They state that as RAD demands a fully functional system between 60 - 120 days it is not well suited to the development of highly complex software. If large applications can be easily decomposed then RAD can work otherwise it should not be used on large projects.

They list the following requirements for RAD :

- A solid methodology
- A central repository for all the information gathered in a business's RAD projects
- CASE tools including a code generator
- Reusable code libraries
- A team approach
- Leadership

The benefits of RAD they list as follows :

⇒ Improved user satisfaction

⇒ Reduced time to delivery

⇒ Lower cost development

⇒ Lower maintenance costs

⇒ Enhanced employee satisfaction.

Sommerville, documents a number of benefits of using rapid prototyping early in the life-cycle [SOMM92]. These include reducing system misunderstandings, clarifying requirements, and the production of a limited system for users for evaluation.

He concludes that effective prototyping increases software quality and can give companies a competitive edge over their competitors.

McLeod Jr. believes RAD is the desired approach to systems development [McLE93]. This is not happening at the moment, he contends, because firms do not have the skilled personnel and have not invested in the computer-based tools essential for the method.

He also talks about using prototyping, as a general technique, in system development and the advantages and disadvantages he perceives in the method are contained in **Table 2.0**.

Table 2.0 - Advantages and Disadvantages of Prototyping [McLE93]

Prototyping Advantages	Prototyping Disadvantages
• Communication between analyst and user are improved.	* Danger of 'Quick and Dirty' development because of time pressure.
• User's needs can more accurately be determined	* Because of their involvement, users may have unrealistic system expectations.
• Implementation is easier because of user involvement in the development.	* Speed of development may result in a poorly designed product.

Hardgrave examined the stages at which prototyping should be used [HARD95]. His study highlighted a number of factors which potential users of prototyping felt were most significant. These factors included :

◊ Requirements Determination - study respondents felt that prototyping could be used when requirements were unclear, expected to change or ambiguous.

◊ Project Size - Large systems should use prototyping as there is a likelihood that requirements will change during development.

◊ Availability of Tools - If suitable tools are available to, say, convert the prototype to the finished product, then this may influence the decision.

◊ Examining Feasibility - prototyping can be used to reduce risk and allow experimentation before full commitment is given to the system.

◊ User Involvement - Prototyping requires significant user involvement and if this is available then prototyping can be used to encourage users to 'buy-into' the system at an early stage. However, unnecessary involvement of users can increase development time.

To conclude this section, the advantages and disadvantages of RAD, as reported in the analysis, are contained in **Table 2.1**.

Table 2.1 Advantages and Disadvantages of RAD (from study analysis)

Advantages	**Disadvantages**
Ease of Implementation	Potential for poor quality systems
Improved User Satisfaction	Need more experienced development staff
Shorter time-to market	Strong project management and control required
Increased system quality	Need for documented standards and procedures

It's not enough, however, for a company to adopt RAD and expect to achieve all the benefits claimed for it; it must change its development process accordingly.

With this in mind and to ensure quality in RAD projects the Dynamic Systems Development Method (DSDM) Consortium was established [DSDM95].

2.3 Dynamic Systems Development Method - A RAD Standard

The Dynamic Systems Development Method (DSDM) was created in February 1995. It was the result of the work of a number of organisations who formed a consortium to examine how projects using Rapid Application Development (RAD) techniques were being implemented. Their objective was to create a method within which RAD techniques could be used but to ensure that quality was built into this approach.

DSDM uses prototyping techniques to ensure the frequent delivery of software products during development. These products are delivered within fixed timescales known as 'timeboxes'. Users therefore receive incremental versions of the finished system. By developing in this way, the DSDM consortium believe that users can continually provide feedback during development and are more likely to receive a system with which they are satisfied. Unlike traditional approaches which attempt to indicate how long it will take to complete a fixed amount of functionality, DSDM estimates are a statement of what will be delivered within a given timebox.

The DSDM authors believe that fundamentally, **it is easier to calculate how much can be done by a certain time than to calculate how long it takes to do something.**

2.4 DSDM - What the Method Contains

DSDM attempts to address the failure of software to meet end-user expectations.

A basic assumption of DSDM is that nothing is built perfectly first time, but that 80% of the solution can be produced in 20% of the time. The Consortium believe that traditional development methods such as the 'Waterfall' model suffer from the fact that requirements must be frozen early in the development and that the wait for one stage to finish before continuing to the next slows development.

DSDM purports to combat this by the use of its iterative approach. This means that the current phase need be completed only enough to proceed to the next step with the flexibility, within the method, to return to the previous step when necessary.

DSDM offers a full software development life-cycle. It provides a framework within which all the individual RAD tools offered by vendors can reside and is, therefore, not vendor-specific.

2.5 Principles of DSDM

The principles on which the method is based are as follows :

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) Iterative and incremental development is essential.
6) All amendments during development are reversible.
7) System requirements are baselined at a high-level.
8) Testing is integrated throughout the life-cycle.
9) All relevant staff must co-operate during development.

2.5.1 Applications Suited to the DSDM Approach

The DSDM consortium has attempted to identify application areas in which it believes DSDM could profitably be used.

These are application areas which:

- Involve systems, where the user interface is of prime importance.
- Possess a well-defined user group.
- Are not computationally complex.
- If large, are capable of sub-division into smaller components.
- Are time critical
- Possess requirements which are 'fuzzy' or not clearly defined.

2.5.2 Applications Unsuited to the DSDM Approach

The consortium believes that the following applications may not be suited to the use of DSDM :

- Applications where functional requirements have to be fully specified before any programs are written.
- Real-Time applications
- Safety-Critical applications.

2.5.3 DSDM Implementation - Critical Success Factors

The factors critical for success in DSDM projects are contained in **Table 2.2.**

TABLE 2.2 - DSDM CRITICAL SUCCESS FACTORS

	DSDM Critical Success Factors
1	The commitment of Senior User Management to provide significant end-user involvement.
2	Easy access by developers to end-users.
3	The stability of the development team.
4	Highly skilled developers in technical and business terms.
5	The decision-making powers of the users and developers.
6	The importance of Project Control.
7	Development team size.
8	A supportive commercial relationship between developers and users.
9	Development technology suitable for use with DSDM.

2.6 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 2.0**.

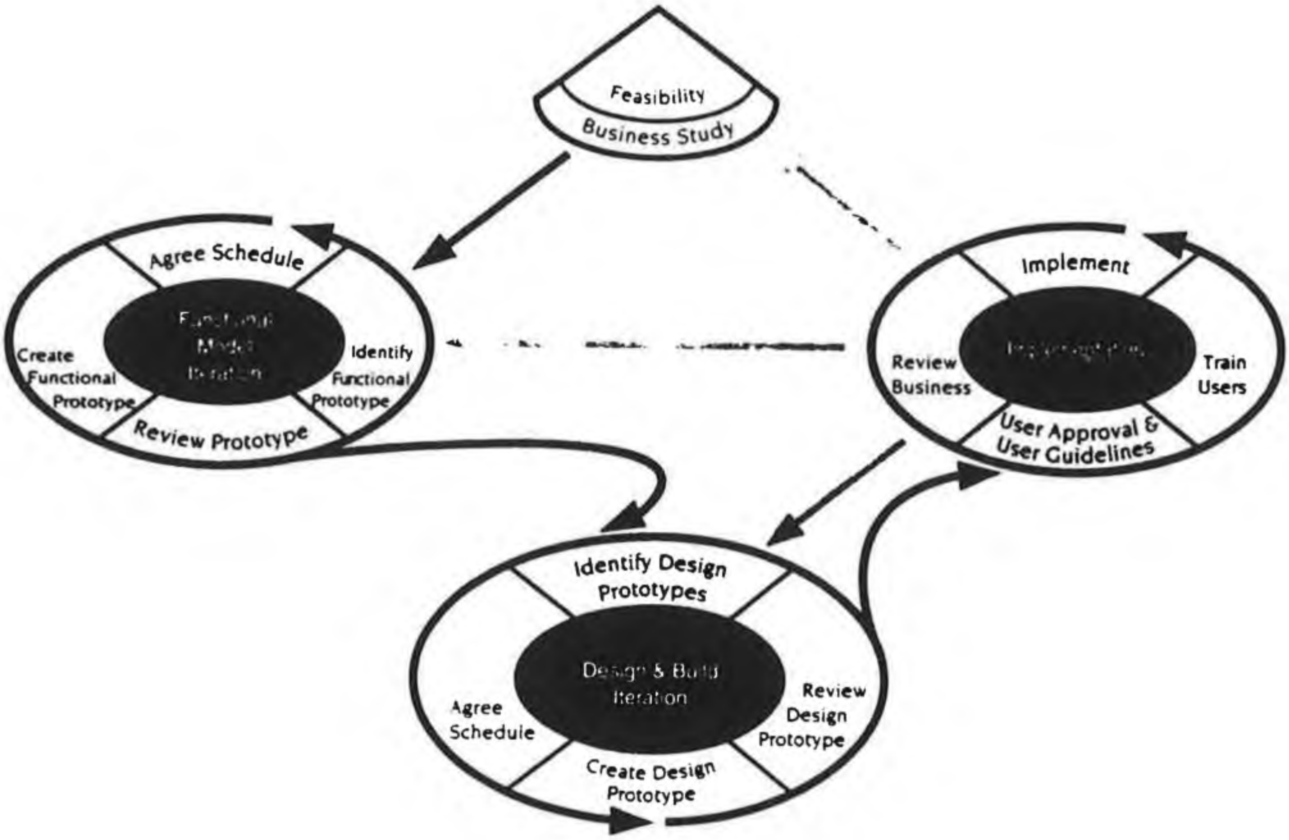

Figure 2.0 - DSDM Life-Cycle

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 method authors state that there should be a maximum of three iterations of each of the prototyping cycles. The final phase, the implementation phase, is the handover to users which will normally be accompanied by a project review.

2.6.1 Prototyping within DSDM

Prototypes in DSDM may focus on :

- Business factors, such as functionality
- Usability factors, such as the User Interface
- Performance and Capacity factors and
- Capability factors (e.g. testing a particular design approach).

Each prototyping cycle contains four stages as outlined in **Figure 2.1:**

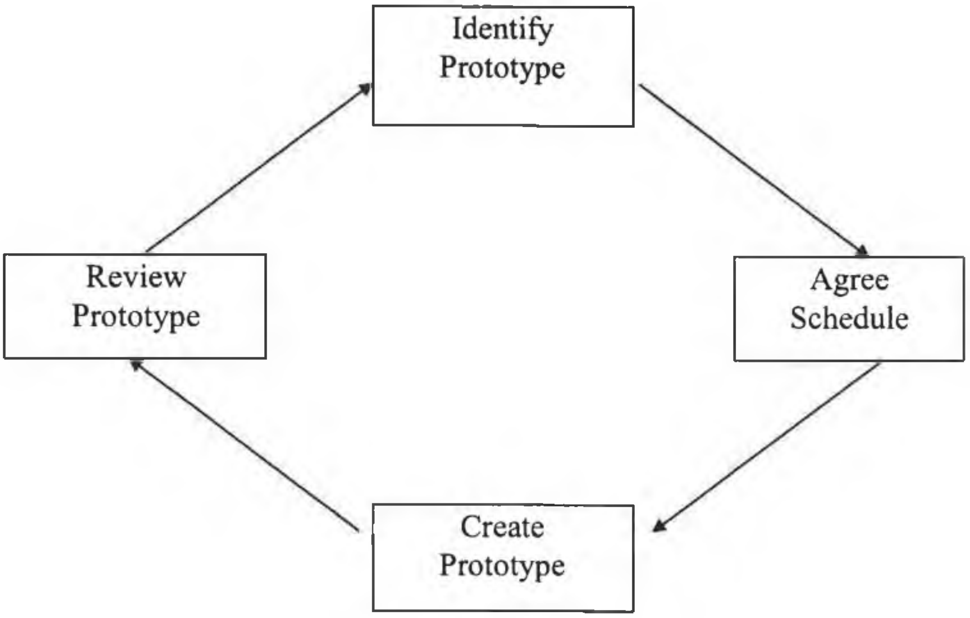

Figure 2.1 - DSDM Prototyping Cycle

1. Identify Prototype

Before building a prototype, the functions to be prototyped must be defined.

2. Agree Schedule

It is vital that a limit be set on the time spent on each prototype. This ensures that the prioritised functionality is developed first.

3. Create Prototype

Prototypes are usually developed in conjunction with users and this is especially important where the user interface or the business functions are being prototyped.

4. Review Prototype

Each prototype should be reviewed by both developers and users.

2.7 Quality Issues in DSDM

2.7.1 Quality Control

Quality Control in DSDM is practised through :

- Product Inspections and Reviews
- Dynamic testing of products and prototypes
- Reviews e.g. of prototypes
- Static code analysis.

Quality Assurance - The consortium suggest that every DSDM project should have an accompanying Quality Plan. QA in traditional life-cycle projects focuses on all the individual stages in the development ensuring that the quality of outputs from one stage is sufficiently high to permit progression to the next stage.

2.7.2 DSDM and the CMM

Regarding the CMM the Consortium believe that introducing DSDM into an organisation can help the organisation achieve maturity level 2 [PAUL93].

This will move the organisation from one which develops software in an ad-hoc fashion to one which achieves repeatability where it is capable of repeating project successes with similar applications.

2.8 Introducing DSDM into an Organisation

There are a number of issues to consider before DSDM can be used within an organisation for a particular project :

- How are the projects currently staffed? Is rigid specialisation present? e.g. programmers only program and carry out no analysis tasks; Analysts never write any code etc.
- Are the project managers empowered? Do they feel they have the power to make decisions?
- Is the current working environment controlled by regulations or consensus?
- Are developers flexible with regard to changes in working practices?
- Can staff relocate on a project-by-project basis?

- Are facilities available for Joint Application Development (JAD) sessions?
- Will operations staff be capable of responding quickly to system requests from the development team?
- Does the development environment allow for prototyping with users?

If any of the above issues present a problem then ways to surmount them should be considered. The presence of a DSDM champion within the organisation will be a significant help in this regard.

2.9 DSDM - A Way Forward for RAD?

It is clear that there are fears abroad that RAD is a return to the bad old days of unstructured development. As such there is a definite need for a methodology which will allow quality to be incorporated into RAD projects. DSDM attempts to counteract this through :

- Inspections, Reviews and Walkthroughs
- Demonstrations to user groups
- Testing (Static and Dynamic Analysis of code).

Testing in DSDM is conducted at every stage of the development process.

While this is very desirable and necessary in an iterative development environment, on its own it is insufficient to guarantee the quality of the finished product. Though the consortium themselves admit, 'testing can never prove that a system works', very little time is devoted by them to the use of alternative measures of assuring the quality of the developed product. They state that all DSDM products can be verified using techniques such as, static analysis, inspections and reviews, but decline to comment on how these techniques should be used and how the results of these activities should be handled.

A large body of work [including FAGA76, FAGA86, WELL93, DAVI94, ACKE89, FREW86] exists which claims that inspections and reviews are superior to testing at discovering errors. However, in DSDM's defence, a lot of this work is based on waterfall approaches where testing is confined to an activity that occurs after requirements have been defined, design has been completed and coding undertaken.

Different results may well occur if testing is integrated at all life-cycle stages.

Nonetheless, the importance of reviews and inspections cannot be dismissed.

The consortium state that there are two key documents for which a formal quality inspection is essential, the Business Area Definition and the Prioritised Functions. However, inspections should at a minimum also be carried out on the Functional Prototypes and the Design Prototypes. The composition of the DSDM team should assist the inspection process. Weller shows in tabular form the benefits inspections have had in his company in **Table 2.3** [WELL93].

TABLE 2.3 - Summary of Inspection Data 1990 to 1992 (from WELL93)

Data Category	1990	1991	1992
Code-Inspection Meetings	1,500	2,431	2,823
Document-Inspection Meetings (anything other than code inspection)	54	257	348
Design-document pages inspected	1,194	5,419	6,870
Defects removed	2,205	3,703	5,649

Any DSDM development team must be multi-skilled with team members having analysis/design and coding skills. Furthermore, users must be on hand to evaluate the user interface elements of any development prototypes. With such strength in depth within the team, formal inspections will be rewarding at every DSDM phase.

When introducing inspection techniques in small companies code inspections could be introduced first and when the benefits have been illustrated, expanded to design inspections. The rigorous use of inspections will also reduce the testing burden on the company. Also in a small company like this, the inspection process can be less bureaucratic. However, it is important that inspection findings are documented and acted upon. Concerns about the length of time that this process will take, with consequent delays to timebox deliverables will be allayed because of the reduction in rework that will emanate from the application of inspections. Ultimately, this can only be proven through the collection of appropriate metrics. Design and Code reviews should also be used as quality indicators. Findings from the PSP show that design and code reviews are superior at detecting errors than is testing.

The reviews that are specified in DSDM are mainly concerned with checking requirements issues with users, with the consortium stating that 'reviews should be short and informal within the development team'. However, design and code reviewing should be conducted by each developer with the results documented. These results, again, can be amassed to reflect process application and effectiveness. Furthermore, as review skills improve, reviews and inspections could replace testing as quality control mechanisms in the early DSDM development phases.

Montgomery, illustrates how the Software Engineering Laboratory (SEL) uses testing as a QA measure and design and code reviews as techniques to ensure requirements are met and code does what it is supposed to do [MONT95]. This would be an improved approach to adopt with DSDM.

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. The success of the quality control procedures depend on the rigour with which they are implemented, the availability of tool support and the development team's capacity to use the tools. While developer lack of familiarity with the tools may introduce an element of risk into the project their absence will certainly slow the time-to-delivery. Also, because of the iterative nature of DSDM development, tools such as those which provide 'capture/playback' testing facilities are imperative as testing must occur throughout the life-cycle. Also CASE tools with code generation facilities are beneficial.

With regard to Quality Assurance practices, DSDM does not define the QA activities it expects users to invoke. However, it does state that every DSDM project should have an accompanying Quality Plan that states how quality and standards are to be enforced. The difficulty that arises here is that the development of the plan for each project will command a time overhead which could impact the delivery schedule. Consequently, the Quality Plan could be the initial basis for document reuse within DSDM projects.

Organisations could derive a template based on the quality factors, referred to in the DSDM document, such as, is sufficient user involvement present?, are priorities being adhered to?, are timeboxes being met?, which require assurance and then insert the appropriate elements based on the particular system being developed.

A section of the DSDM manual discusses the method with relation to the Capability Maturity Model (CMM). As documented previously in this study, the CMM assesses the maturity of the software development process within an organisation.

In the DSDM manual it states 'The DSDM Consortium believe that introducing DSDM into an organisation can help the organisation achieve process maturity level two'. Process maturity level two, 'Repeatable', describes a process where basic management controls exist to track cost, schedule and functionality and the discipline exists within the process to repeat previous project successes.

This, at first, appears to be a fairly modest target for a methodology which is defining a life-cycle approach incorporating a number of quality measures.

However, there are two factors which clearly illustrate why the Consortium's target of level two attainment is correct.

Firstly, DSDM is a method for use in RAD projects. As stated previously not every project is suitable for RAD development, so the approach could not become an organisation wide development standard unless every project the organisation developed used RAD techniques!

Secondly, the CMM framework describes the capability of an organisation and as such requires substantial organisation and management structures and practices which go beyond the mere adoption of the development framework which is DSDM.

To progress to level 3 would, therefore, require organisational changes.

In conclusion, then, DSDM is a very useful addition to organisations wishing to adhere to the CMM criteria.

2.9.1 Other Factors which may Influence Development

There are other factors which have not been directly referred to in the published articles and which affect the efficacy of DSDM.

Because user involvement is fundamental to the success of RAD, companies writing packaged software or bespoke software for another company may have difficulties with user involvement in development.

Obviously, companies writing packaged software are writing for a whole class of users who may have different applications for the package. However, the project sponsors are likely to be based internally and so can act as users. The difficulty of not working with the actual users of the product may mean that business objectives are not met and users receive a system which does not actually meet their needs.

Problems in this regard may be countered by Beta testing the product at a number of selected sites. In fact using RAD, a version of the product with perhaps limited functionality could be released more quickly, produce early feedback and the comments made fed into the next system prototype. Used in this way RAD can assist in producing a system that is more likely to meet users needs much more quickly than using traditional approaches.

Writing bespoke software raises other issues. If the developers and users are physically separated, say working at their own individual company's premises, then development may be slowed awaiting the confirmation of decisions etc..

Also, in this scenario users, particularly, are likely to get drawn into their own day-to-day work with the result being reduced commitment to the project. Collocation of developers/users will assist greatly in this regard. Users may not be required for 100% of the time but should be available when needed on the project.

Because of this it may be best if developers were actually sited at the users premises during bespoke developments using DSDM. The manual addresses this saying 'Risks can also be mitigated by seconding and locating developers where the users are situated. This is best done physically but....it has been done using groupware and video/e-mail conferencing facilities to create a virtual location.' In this way the collocation requirement is being met but users can execute their day-to-day duties when not working on the project. To succeed, this requires the support of user management. Without it the project is at risk and delivery time will be extended.

Component reuse is encouraged in DSDM, nonetheless, the manual suggests there may be a risk that 'the DSDM team may start building components for future reuse, incurring development costs not related to the DSDM project objectives'. The Consortium believe that the 'costs in creating and supporting reusable materials...should be regarded as corporate investments in infrastructure and assets..'. They suggest that after delivery of a RAD project developers may be involved in 'enhancing and re-engineering components that were identified during the project as potentially reusable.' Though RAD components are not being specifically designed for reuse there is also a danger that they may be unmaintainable.

The manual is somewhat vague on the question of maintenance. It states that 'systems with poor maintainability :

- take more resources in maintenance
- take longer to change
- are more likely to introduce further errors with change and be unreliable
- will cost more to maintain'.

While re-engineering reuse components was considered to be a corporate cost then maintenance costs certainly relate to a project. The manual states 'It is therefore important that the system as a whole and all its components are engineered to be maintainable from the start...'. It goes on to say that with RAD 'An environment is created where a small team works...towards...providing maximum business benefit'. The section concludes by saying 'One of the principal advantages of DSDM is that it explicitly deals with these dangers'.

How DSDM actually deals with these dangers is that decisions regarding maintainability are taken by senior user management. The manual outlines the three possible choices of business objective to cover maintainability :

- maintainability is a required attribute of the initially delivered system
- A short-term tactical solution - earliest delivery is paramount; the system will be replaced/rewritten before maintenance costs are a problem.
- deliver first - re-engineer later - time-to-market is important and re-engineering to provide maintainability will occur after implementation.

Later on the manual comments that 'DSDM does not ensure maintainability by itself. [Maintainability] is made possible by a combination of...

- tools
- people
- documentation
- good practice guidelines'.

All of these factors are indeed relevant not just to DSDM projects but to every development approach.

As can be seen from the above the DSDM manual is slightly ambiguous regarding maintenance. The maintainability of the delivered system is not guaranteed by the method but is based on the decision taken by the project sponsors.

Another factor is in regard to integration of software. With RAD there is the temptation to build stand-alone systems which can be delivered within the chosen timescale rather than ones which will integrate with existing systems but cannot be delivered within the allocated timeframe. Such systems may not easily interface with current ones leaving a substantial engineering task in order for them to do so. Again with DSDM this will be a management decision regarding the type of system they wish to deliver. Once again though the cost of integrating the system into the current environment post-delivery must be allocated to the project itself.

2.10 DSDM - The Benefits

The first benefit that can be attributed to DSDM is that it provides a quality-oriented approach for RAD development. Some of the fears raised related to the absence of a methodology for RAD.

A study of the DSDM manual will show that it is a well-thought out approach and covers all of the issues raised in the articles and addresses the perceived disadvantages of RAD.

The second benefit of RAD, as was extensively argued in section 2.2, was of how well it assists in ensuring user satisfaction with the delivered system.

This is true in a number of ways, such as, ensuring the 'right' system is built, helping to clarify requirements, increasing user acceptance by having them involved in development and in promoting developer/user communication. DSDM underpins these benefits. As stated on a number of occasions a fundamental assumption of DSDM is to ensure that business objectives take precedence over everything in a DSDM project. By providing a documented life-cycle it supplies a framework in which prototypes, which help clarify requirements, can be developed.

The methodology insists on some form of collocation of developers and users, thus helping team communication. Other contributions in this area are the definition of team structures contained in the manual which if followed will supply the necessary experience from the user and development side. Also the way the life-cycle is documented coupled with the team structures will significantly contribute to good communication. Furthermore, the use of software tools which provide diagrammatic and graphical representation of the system, encouraged in DSDM, will aid communication even more.

The third major benefit claimed for RAD is that of increased productivity and reduced time-to-market. The use of advanced software tools including code generation, the ability to produce prototypes and the emphasis on rigid adherence to timeboxes within DSDM again help underpin these benefits.

Another means to reduce time-to market is through software reuse. The method itself doesn't contribute significantly in this regard. While it promotes reuse where components are available, it suggests designing for reuse may itself reduce time-to-delivery.

Of the other factors listed and contained within the articles, the final element referred to the quality of the finished product. A majority of the authors who commented on this area felt quality may suffer using RAD, however, some felt quality may be improved. This improvement, it was felt, would come from increased user involvement and the higher chance of meeting business objectives. Proponents believe the delivered system would be more likely to possess a 'fitness for purpose' attribute.

2.11 Summary

DSDM with its emphasis on user involvement within the development process and the emphasis on building the 'right' system contributes to the likelihood of user satisfaction being achieved. The provision of a prototyping methodology, the inclusion of quality mechanisms, the emphasis on testing throughout the life-cycle, the personnel mix and the guidance on software tool support all contribute to improving the quality of the delivered product and to ensure 'fitness for purpose'.

DSDM has great potential for small companies in that:

- It provides a defined and documented life-cycle
- It can assist in reducing development time; crucial for companies operating on small margins or where getting to market first is vital
- It is best suited to small teams.

The third factor here is particularly relevant to small companies as in DSDM-based projects developers and users sometimes play more than one role. It is common in small companies for employees to have less structured roles. For example, project managers may act as programmers and systems analysts may write technical documentation. These factors make DSDM potentially a very beneficial approach for small companies.

Chapter 3 – Process Improvement using the Personal Software Process (PSP)

3.0 Introduction

The purpose of this chapter is to examine the Personal Software Process (PSP), which was designed by Watts Humphrey of the Software Engineering Institute [HUMP95]. It describes, in detail the components of the PSP and the approaches and documentation associated with the PSP. The approaches contained within the PSP are then analysed by relating them to other relevant studies in the area. The chapter then concludes with a summary of the PSP and its relevance to small development organisations.

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. The objective, of the PSP, is to make the individual a better software engineer. It is essentially a bottom-up approach where individuals manage and assess their own work as opposed to the organisation-wide approach of, for example, the Capability Maturity Model [PAUL93]. As such the PSP 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 to assist in improving performance at the individual level. It provides historical data which helps you measure your performance, your work patterns and practices. By examining these and using the PSP framework Humphrey believes the developer can :

- Plan better
- Track performance
- Measure product quality
- Improve productivity
- Make more accurate estimates
- Reduce defects.

Examples of the forms used in the PSP are contained in **Appendix C**.

3.2 PSP Improvement Phases

The Evolution of the various PSP phases is illustrated in **Figure 3.0.**

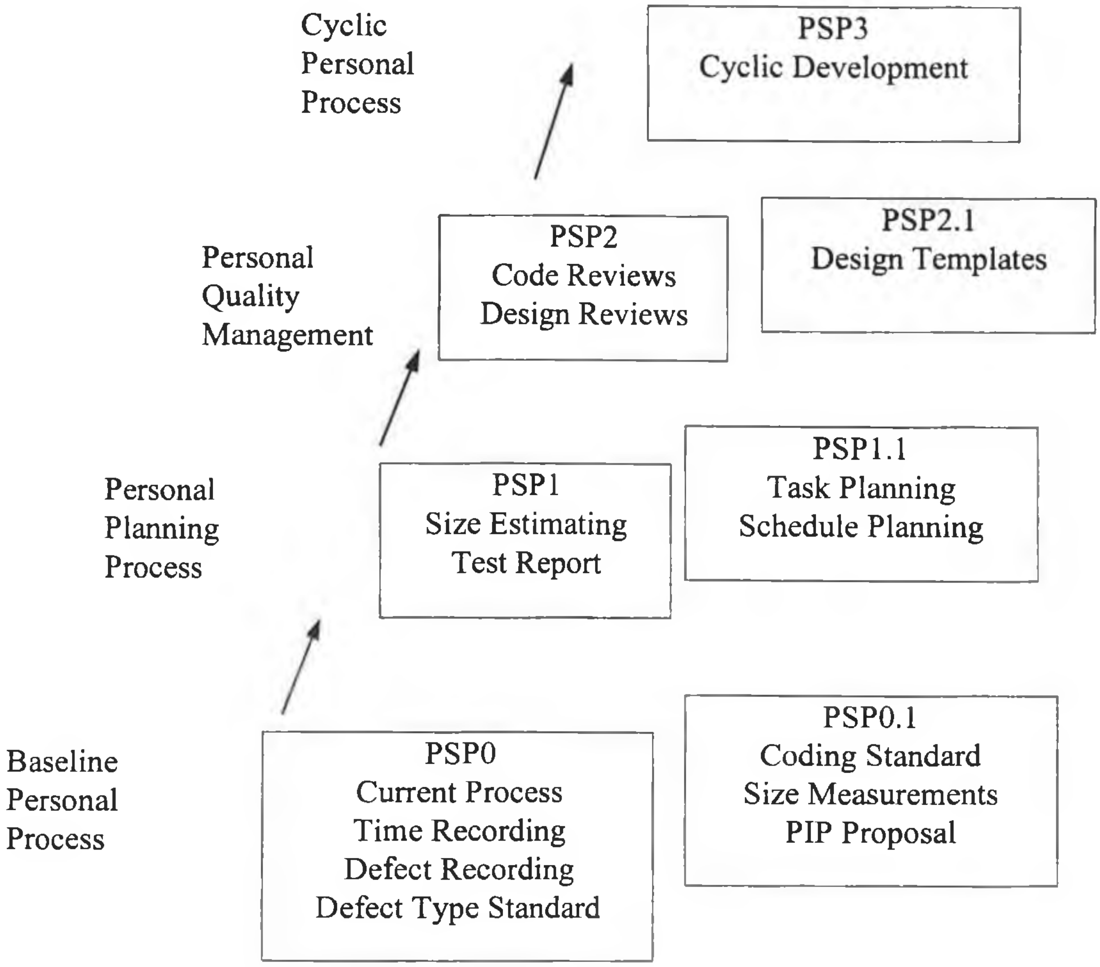

Figure 3.0 - The PSP Evolution

3.3 The Baseline Personal Process (PSP0)

The principal objective of PSP0 is to provide a framework for gathering your own initial process data. Each PSP phase is accompanied by scripts, logs and summaries, which in effect produce a defined process.

PSP0 has three elements - the Planning phase, the Development phase and the Postmortem phase. Each of these elements is accompanied by a script.

There is also a Process script to ensure that the individual phases are being executed correctly. Each of the scripts contain Entry and Exit criteria and individual phase elements. PSP0 has two measures :

- The time spent per phase and
- The defects found per phase.

The ***Time Recording Log*** (Appendix C) is used to document the time spent on each phase. The benefit of the time recording log is that not only does it show you how your development time is distributed (i.e. the time spent per phase) but the frequency and duration of interruptions.

The ***Defect Recording Log*** (Appendix C) is used to record defects as they arise in each phase. Each defect is allocated a number, a type e.g. syntax, interface etc., and the phases at which the defect was injected and removed are entered. Another important factor to record is the time taken to fix the defect. It is also possible to record defects introduced while fixing another defect using the 'Fix Defect' column.

The ***PSP0 Project Plan Summary*** (Appendix C) requires you firstly, to document your estimated time for the development. Then, on completion, you enter the actual time spent in each phase and the phases in which defects were injected and removed. The figures will be taken from the Time Recording Log and the Defect Recording Log respectively. To date figures are included to assist in measuring progress.

3.3.1 PSP0.1

PSP0.1 extends PSP0 by the inclusion of additional planning and size measurement details. Planning is the first step in the PSP. The plan defines how the work is to be done and allows for comparisons with actual performance. The second element in planning software projects is measuring software size. If you can estimate the size of the product you plan to build, you can then make better judgements about the amount of work required to build it. Size can be measured by counting Lines of Code (LOC), Function Points (FPs), Objects or some other suitable unit. Lines of Code (LOC) are usually based on program source code and normally exclude comments and blank lines.

Function Points (FPs) are derived from counting certain parameters e.g. number of user inputs, number of external interfaces etc. and applying a weighting complexity factor to these parameters. These weighting factors may be adjusted based on other system related factors, such as, whether the code is to be designed to be reusable or if performance is critical. Another important element of PSP0.1 is the Process Improvement Proposal. Which provides a way of recording process problems and improvement ideas. This form can then act as input for later process improvements.

3.4 PSP1

3.4.1 Proxy-based Estimating

In order to assist with size estimation, Humphrey proposes the use of Proxy-Based estimating. Because few people can judge accurately how many LOC it will take to meet a software requirement there is then a need for a proxy to be used. 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. Objects or functions fulfil the proxy requirements particularly well. Humphrey suggests using the **PROBE** (***PRO***xy-***B***ased ***E***stimating) method with the PSP.

Table 3.0 shows a sample proxy table for C++ objects.

TABLE 3.0 - Object Category Size in LOC per method (C++) (from HUMP95)

C++ Object Size in LOC Per Method					
Category	Very Small	Small	Medium	Large	Very Large
Calculation	2.34	5.13	11.25	24.66	54.04
Data	2.60	2.47	8.84	16.31	30.09
I/O	9.01	12.06	16.15	21.62	28.93
Logic	7.55	10.98	15.98	23.25	33.83
Set-up	3.88	5.04	6.56	8.53	11.09
Text	3.75	8.00	17.07	36.41	77.66

3.4.2 Resource and Schedule Estimating

Having made a size estimate, you now need to decide the time the work will take, assess the accuracy of this estimate and generate a development schedule.

When estimating the program size you used your historical size data as input to that process. Consequently, you use your historical productivity figures and the data relating to your resources available as input to your resource estimates. Combining these estimates produces the schedule.

3.4.3 Measurements in the PSP

Measuring your process allows you to understand how it works and look at ways it can be improved.

1. Product Measures

These generally refer to the volume of product produced and include LOC, pages of documentation, numbers of screens/files etc..

2. Process Measures

These can include such as, number of defects removed in test, number of changes made to requirements, number of defects injected per phase. You may also use cycle time (time taken to complete a project) as a measure.

3. Resource Measures

While productivity measures are of use, equally important is the breakdown of time spent in development.

3.5 PSP2

3.5.1 Design and Code Reviews

Humphrey believes, that by doing design and code reviews you will see greater improvements in your own personal software process than through any other change you may make. The principal review methods used in software development are inspections, walk-throughs, and personal reviews.

An ***inspection*** is a structured procedure for allowing a team of people to review a software product.

Each participant in an inspection has a defined role and the material to be inspected is distributed to the participants in advance of the inspection meeting.

A ***walk-through*** is less formal than an inspection and usually takes the form of a presentation by the programmer/designer with the presenter going through, step-by-step, how the software will perform.

A ***personal review*** occurs where you examine your own software products. The objective is to find and fix as many defects as possible prior to inspection, compilation or test.

Apart from code, design and requirements documents, test plans, user documentation etc. can also benefit from the review process.

PSP data gathered by Humphrey show that extra time spent in reviews is more than compensated for by reduced time in compilation and testing.

3.5.2 Software Quality Management

There are two elements to software quality management :

- Product quality, and
- Process quality.

3.5.2.1 Product Quality

The first element of product quality is that the software product must meet the users' requirements at a time when the users need them. Secondly, the software product must work. If the product is riddled with defects then it will not be used. Thirdly, the software product must be capable of handling the longer-term quality issues, such as, maintainability, portability, usability etc.

The manifestation of poor quality software is the concentration in the development process of finding and fixing defects. By reducing defects in the development process, focus can then be placed on the other aspects of software quality.

3.5.2.2 Process Quality

A quality process should meet the needs of its users, i.e. software engineers.

The hallmark of a good software process is that it produces good software products, consistently.

3.6 PSP3

3.6.1 Scaling Up the Personal Software Process

As the size of software systems increase, how can the PSP be adapted for use on these larger systems? Large-scale systems must be decomposed into manageable subprocesses. By refining these, and defining methods, you will be able to create a vocabulary of individual processes. These known and repeatable methods can then be used in scaling up our processes.

3.6.2 The PSP3 Approach

The role of PSP3 is an example of the personal process for large-scale software development. The key element in PSP3 is the Cyclic Development Process.
Each cycle is essentially a PSP2.1 process that produces a part of the product.
During the cycles the reviews and tests are as complete as possible.
Figure 3.1 shows the cyclic development process associated with PSP3.

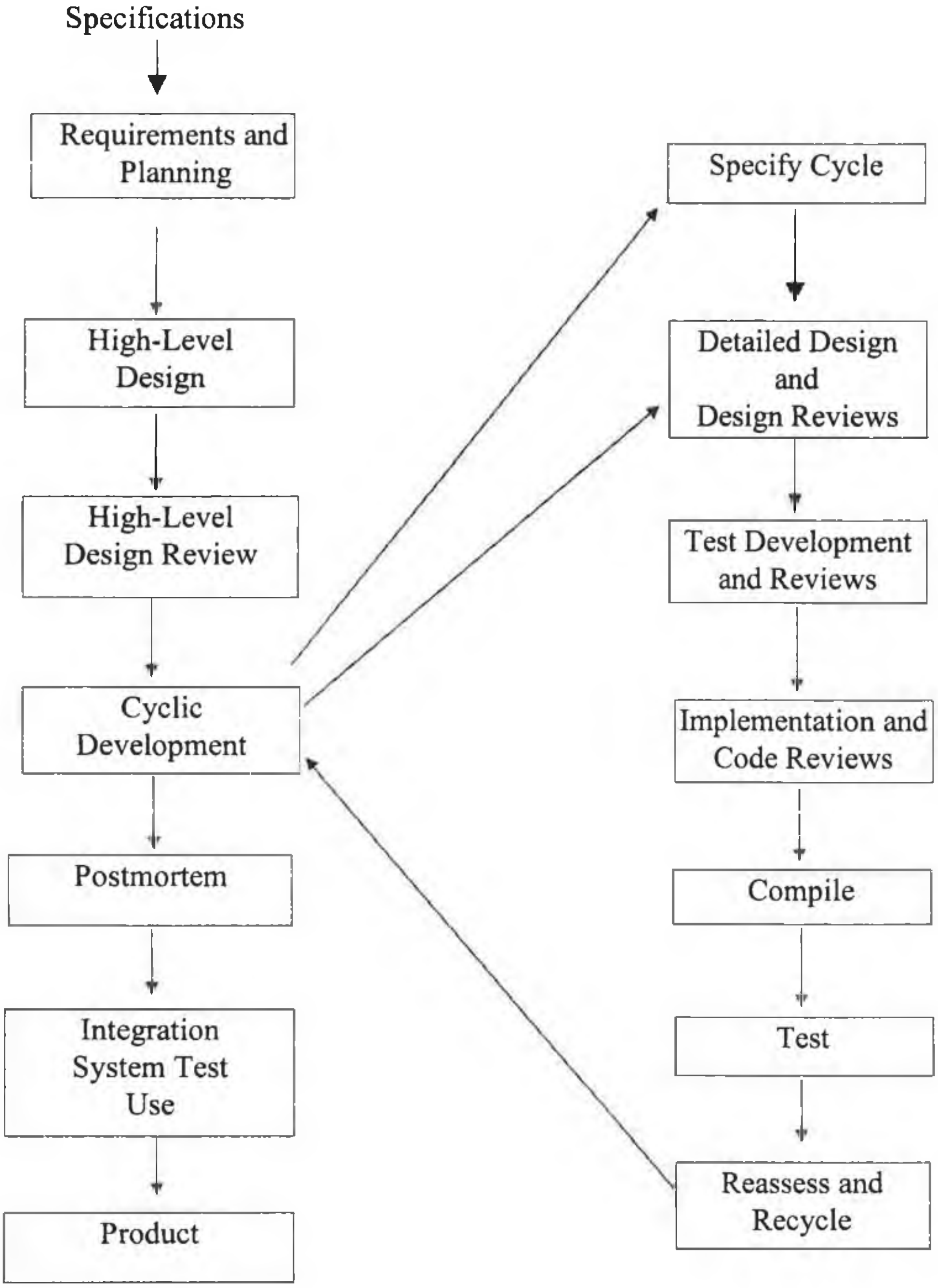
Specifications
Requirements and Planning
High-Level Design
High-Level Design Review
Cyclic Development
Postmortem
Integration System Test Use
Product
Specify Cycle
Detailed Design and Design Reviews
Test Development and Reviews
Implementation and Code Reviews
Compile
Test
Reassess and Recycle

Figure 3.1 - PSP3 Process

3.7 Using the Personal Software Process

It will be easier to use the PSP if the organisation supports your improvement efforts. If you are the only person using it, you will find difficulty maintaining the disciplines required.

If it is to be introduced into an organisation then there must be commitment from senior management, time allocated to it, support given and a schedule for implementation.

3.8 Personal Software Process - An Analysis

Having looked in detail at the PSP it is instructive to examine what others have to say about the approaches and techniques the method proposes.

Cusumano, in his paper on the 'Software Factory', states that a 'software factory' should have measures and controls for productivity and quality. One Japanese 'software factory', he studied, set two goals: firstly, the achievement of productivity and reliability improvement through process standardisation and control, and secondly, the transformation of software from an unstructured service to a product with a guaranteed level of quality [CUSU89]. These approaches, he claims, helped the Japanese produce 50 percent fewer bugs per KLOC and required less maintenance than US projects.

Grady, discusses, in his study, ways of finding program defects, categorising them and subsequently analysing them [GRAD93]. His approach was to use inspections, code coverage, and complexity measures along with testing to uncover errors. The lessons learned were that changes, prompted by metric results, were the easiest ones for organisations to accept and implement. Reducing rework was possible using the proven software engineering methods. Analysis of defects allows you to focus process improvement decisions on the most important problems and the monitoring and measurement of product quality is the single most important performance measure.

Wohlwend and Rosenbaum, in a study of their company's organisations, carried out an evaluation to see where improvements in software development capability could be made [WOHL94]. Their improvement efforts included focusing on tracking the deliverable size and efforts on current projects, and gathering data about code and testing errors. These are areas which are covered particularly by PSP1 and PSP2.

Sharp, believes that the one dominant factor in determining software quality is how well the project is managed [JOCH95]. He cautions, however, that reliable software often has fewer features and takes longer to produce. One of the companies Joch, spoke to talks about the exorbitant cost of fixing a defect when software is in use compared to fixing it if found during coding ($1000 Vs. $1) [JOCH95].
These companies have managed to reduce their error rates substantially by using code reviews both for original code and, importantly, for subsequently amended code.

Montgomery, states that though creating reliable software is difficult it is a product of the management of processes methods and tools [MONT95]. Commenting on NASA's Software Engineering Laboratory (SEL) he states that their objective is to produce error-free software. Their approach means that code reading and peer reviews are used to ensure that the code does what it's supposed to do and testing is therefore concerned with quality assessment.

Kitchenham and Pfleeger, in their analysis of quality, show how the degrees of quality are significant stating that errors in a word-processing package would likely not be acceptable in a safety-critical environment such as a nuclear-power plant [KITC96].
Measurement, they say, must be able to assess how process quality affects product quality. Their paper also contains responses from IEEE Software board members to quality related questions. Roger Pressman (on 'selling quality methods to management') believes you should approach it on the basis of cost savings. Cost data connected to defects should be collected and the cost of quality determined as selling factors, he believes.
Ways of measuring the above are contained within the PSP through its standard defect measures, cost of quality measures, and the derived defect and quality measures.

Also in Kitchenham and Pfleeger's paper, Larry Druffel, when asked about generating management and employee support for introducing quality practices, believes that these individuals must be aware of dissatisfaction with current practices before they will be motivated to make changes.

However, if they have not measured their current process, then they will not be aware of what state the process is in.

The PSP through its baseline (PSP0) process will allow initial measurements to be made and therefore provide an assessment of current proficiency. On the economic costs of software quality, Pressman advocates shifting costs, into areas such as reviews, where higher quality can be produced.

Dromey, warns that the widely-held belief that a quality product depends on a quality process can mislead if the focus on process comes at the expense of constructing, refining and using adequate quality models [DROM96]. If a quality product does indeed depend on a quality process, he states that we should ensure that the product is developed within a mature, well-defined process. Use of the PSP can provide the path to this goal.

Lindstrom, details how a large-scale development project can be destroyed [LIND93]. Some of the pitfalls he encountered included : inadequate tracking and management of system and software requirements; selection of design, production and test and integration methodologies that were inappropriate to the development; and a failure to provide a metrics program that would let managers track the progress of software production and test. During the development the company used methods that had previously been successful with small programs. When the system turned out larger than anticipated, the company was unable to scale up its methods to meet this requirement.

This capability of scaling up software engineering practices is well addressed by PSP3.

Another problem the project encountered was the fact that it was ineffectively tracked. The management believed the project to be near completion and reduced resources accordingly. This proved disastrous as the project was in fact entering a critical phase when all resources were needed.

The PSP's Schedule Planning capability coupled with is concept of Earned Value provides exactly the sort of information needed to see how closely the project is on schedule and would help to avoid the sort of scenario outlined above.

As Lindstrom says, metrics may not prevent schedule slippage but they allow projects to be monitored and enable timely, corrective action to be taken.

Reporting from a software measurement conference, Burgess outlines how one of the contributors warned of fitting historical results to a new project if those results themselves stem from poorly managed projects [BURG95]. Another contributor highlighted the crucial role of company management in a successful metrics programme. Unsuccessful programmes contained staff who felt the data collected was inaccurate, no feedback was provided and the metrics data was being used surreptitiously for performance appraisal. The dangers of using metrics in this way are highlighted by the PSP.

There are some, however, who challenge the primary quality goal of defect-free software. Yourdon, in his paper, introduces the concept of "Good Enough" software, an approach, he believes, can be a key factor in the success of software companies [YOUR95].

He cites the popularity of some of the most successful word processing and spreadsheet packages. Some of this software, it is widely recognised, contain many defects, yet judging by sales figures this software is obviously considered "good enough" by buyers. In conjunction with this he asserts that for many customers how quickly they receive the software can be more important than the number of faults it contains as delayed software, even a defect-free product, may mean a lost business opportunity. While he's not suggesting that safety-critical systems should be other than defect-free he raises the important issues of the time and cost required to produce defect-free products. In some instances, he suggests, these may not be the highest priorities.

Carmel, carried out a survey of companies involved in producing software packages [CARM93]. One of his findings was that developers were aware that users were prepared to accept some level of defects in software. Significantly, the experience of these companies was that market early adopters were more prepared to accept a less-than-perfect product than late adopters; the time-to-market argument again.

Another interesting finding was that companies felt that introducing quality assurance would increase development costs and would have such a detrimental effect on pricing as to reduce overall revenues. The survey also revealed that in the companies questioned the software developers had little experience or training in quality assurance. Interestingly, all the companies surveyed were all small software development environments. Carmel comments that it may not be appropriate to introduce the process formalisms developed for large-scale development into such small companies.

Fagan, believes that the successful management of any process requires planning, measurement and control [FAGA76]. He argues in favour of design and code inspections as a way of reducing defects in programs thereby improving the quality of the product. He claims that the cost of reworking errors in programs becomes higher the later they are reworked in the process, so every effort should be made to find and fix errors as early in the process as possible. As such he believes inspections, which result in finding errors early, reduce the overall rework time and increase productivity. This would concur with the view expressed by Humphrey in the PSP. Indeed Humphrey suggests that "there is some evidence that inspections are as good as, or better at, finding the difficult-to-fix defects than is test".

In his follow-up paper, Fagan, defines a defect as "an instance in which a requirement is not satisfied" [FAGA86]. He discusses the need to inspect requirements and test plans. In every instance, throughout the life-cycle, one of the benefits of carrying out inspections is that they are performed much nearer the point of defect injection than is testing. This, he says, is achieved by inspecting the output of each operation to check that it satisfies the *exit criteria* of the operation. Consequently, this will reduce the find/fix time and reduce overall project costs. This concept of exit criteria is in tandem with what Humphrey proposes in the PSP, for each PSP level.

Weller, discusses his company's successes with software inspections [WELL93].

During the inspection process a number of metrics are collected including: time to prepare for the inspection; time to conduct the inspection and defect data including defect type. This then allows them to execute a defect causal analysis.

The PSP also includes a table of defect types and its author states that as the PSP matures within companies they can then start conducting defect analyses, to see the type of defects being introduced at each stage in the process. These analyses can then assist in defect prevention. Weller also comments on the reluctance of some developers to conduct code inspections prior to testing. He laments that code inspection after unit test is still more popular even though it has been documented that this yields far fewer defects. His company's experience also showed that defect detection rates were lower when they inspected after unit test and, significantly, there were more defects in the product after shipping. The self-convincing nature of following PSP disciplines is important here. By following these disciplines practitioners can see the benefits they produce at first hand as the metrics are collected on route. Weller asserts that the following disadvantages are associated with inspecting after unit test :

- It lowers the motivation of the inspection as developers have false confidence in the 'tested' product
- Following test, the temptation exists to bypass code inspection and proceed to integration test
- Unit testing first reduces the opportunity to save time and resources. A good, mature inspection process producing good results may allow you to bypass unit test and proceed directly to integration test
- Inspections often highlight design defects. Waiting until after unit test may increase rework and, therefore, project costs.

He warns, however, that no amount of inspection can make up for design flaws. The lesson is to get the design right in the first instance. The PSP's emphasis on design verification can help to ensure that the system design is indeed correct.

Davis, in his paper also advocates inspecting code as a much better way of finding errors than testing [DAVI94].

He reports on data that shows that inspections can reduce testing time by 50 to 90 percent. Gilb, quoted in a software quality conference proceedings report, proposes inspections for all documentation as well as code [MYER93]. He believes that documents for a project phase should not be accepted unless they exit their previous phase with no more than a maximum number of remaining defects.

Ackerman et al, discuss inspections and say they are superior to reviews and walkthroughs as a verification process [ACKE89]. They report on data that show that inspections reduce the testing effort substantially and that they are between 2 and 10 times more effective at defect removal than testing. Two important advantages of software inspections, they list, are firstly, that they improve quality by reducing the number of defects in the released product and secondly that they improve productivity by removing defects earlier when they take less time to fix. They also support inspections by stating that it costs 20 times less to remove defects through inspections than through tests and that finding a major defect at inspection takes about one hour compared with about nine hours for testing.

Frewin and Hatton, also comment on the increasing cost of removing a defect the further its discovery is from its source and that the output from each stage must be thoroughly reviewed before progression to the succeeding project stage [FREW86]. They also believe that reviewing is the most cost-effective way of removing defects. Quality management, they state, achieves known and predictable product reliability which enables the software user to plan and schedule more accurately.

3.9 Summary

As has already been illustrated in this study, most organisations have introduced, or wish to introduce software process improvements into their development environments.

For some it has been the adoption of standards. For others it has been the introduction of a metrics program. For many, it has been the use of software inspections and review techniques to isolate errors earlier in the development process.

The quality of the software product is an issue that is facing most 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.

The results detailed by the study have shown the benefits of such process improvements.

Software inspections have helped a number of companies to find and fix errors earlier in the development life-cycle, when it is cheaper to do so. Some have also seen further improvements with less time spent in unit testing as a result of this. Using inspections, some have noted productivity increases with products being released earlier and significantly less rework. For other companies, it has been the introduction of reviews which has led to improvements.

Software metrics are being used more and more 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.

Knowing the type of errors generated and the life-cycle stage at which they are introduced, organisations have then been able to focus on their development weaknesses. Improvement efforts in these areas are in the form of defect prevention methods as opposed to the defect detection approaches of inspections and reviews.

The study has highlighted the role of management in any process improvement programme. Company management must be prepared to devote time and resources to ensure that process improvement measures succeed.

What companies must not do is to use any metrics collected as an instrument in performance appraisals. Should employees have even the vaguest suspicion that this is indeed the case then such a programme is doomed to failure and if the company persists will result in inaccurate metric figures and low staff morale.

While time-to-market emerged for some companies as the primary issue use of the PSP may not be a deterrent in this regard. With its support for verification measures, such as reviews and inspections, more defects may be caught earlier in the process.

As the studies show, use of such techniques can result in productivity increases which in turn can aid in faster product delivery. Continued use of the PSP will also result in a standardised and repeatable process, which can also help in the speedier development of software. The Personal Software Process contains the appropriate mechanisms to assist in implementing all the process improvement factors outlined.
It supports the collection of metrics, details the relationships between product size, development time and productivity and outlines the techniques for removing defects early and implementing defect prevention programmes.

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. However 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. PSP is also a bottom-up approach and is, therefore, more quickly and easily implemented in a small company. In a larger company PSP would have to be used in conjunction with a larger process model as it would be necessary to capture corporate-wide metrics and develop large-team and corporate-based processes.

Taken together, PSP and DSDM offer significant potential for software process improvement in small companies.

Chapter 4 - Assessment of the Software Process within a Small Company

4.0 Introduction

The purpose of this chapter is to assess the state of process maturity in a small software company. Without real data on how small companies operate, it is difficult to determine which process models are appropriate. With this in mind a small software company was chosen to ascertain what approaches it took to developing software, what weaknesses existed in this process and where improvements could be made.

This company assessment is divided into two sections. Firstly, a questionnaire was used to obtain general information on the company's processes; the questionnaire was used with members of staff at different levels in the company.

Secondly, having processed the questionnaire responses, I then proceeded to perform a detailed evaluation of the company. This is covered in the next chapter.

4.1 Software Process Questionnaire

The use of a software process questionnaire, in this way, is an attempt to gauge an understanding of the process and the development environment **(Appendix D)**.

The questionnaire is based on the Software Process Maturity Questionnaire, which has been developed by the Software Engineering Institute as a method to appraise the maturity of software development organisations [SPMQ94].

The SEI's maturity questionnaire identifies key process areas in the development of software. However, some of the areas targeted in the SEI's questionnaire are appropriate for large companies only. In a small development environment, roles are often amalgamated and the necessary bureaucracy often associated with a large organisation is mostly absent. As such it was necessary to tailor the questionnaire to suit the small software house under study.

The approach 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 questionnaire was delivered in a structured format with questions being addressed directly to the interviewee, the responses being documented and any extra relevant information not covered by the questionnaire being captured. The questionnaire was completed by members of staff at all levels of the company, so that a broad range of views and understanding could be achieved. Respondents were asked to reply **'Yes', 'No', 'Don't Know' or 'Does Not Apply'** to a series of questions grouped under a range of headings which covered all aspects of the Software Process. The approach I took to documenting the answers is the same as that used in the SEI maturity questionnaire and these instructions accompany the questionnaire in **Appendix D**.

The headings used in the questionnaire were:

Project Review and Sign-Off
Configuration Management
Software Estimation
Metrics
Education/Training
Project Management
Software Quality Assurance
Requirements Specification
System Design
Implementation
Testing
Operations/Maintenance.

4.2 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%.

Process Area	**% of Target Achieved**
Review and Sign-Off	**30**
Configuration Management	**27**
Estimating	**31**
Metrics	**0**
Education/Training	**32**
Project Management	**46**
Software Quality Assurance	**0**
Requirements Specification	**33**
System Design	**27**
Implementation	**21**
Testing	**13**
Operations/Maintenance	**27**

These results are represented graphically in Figure 4.0.

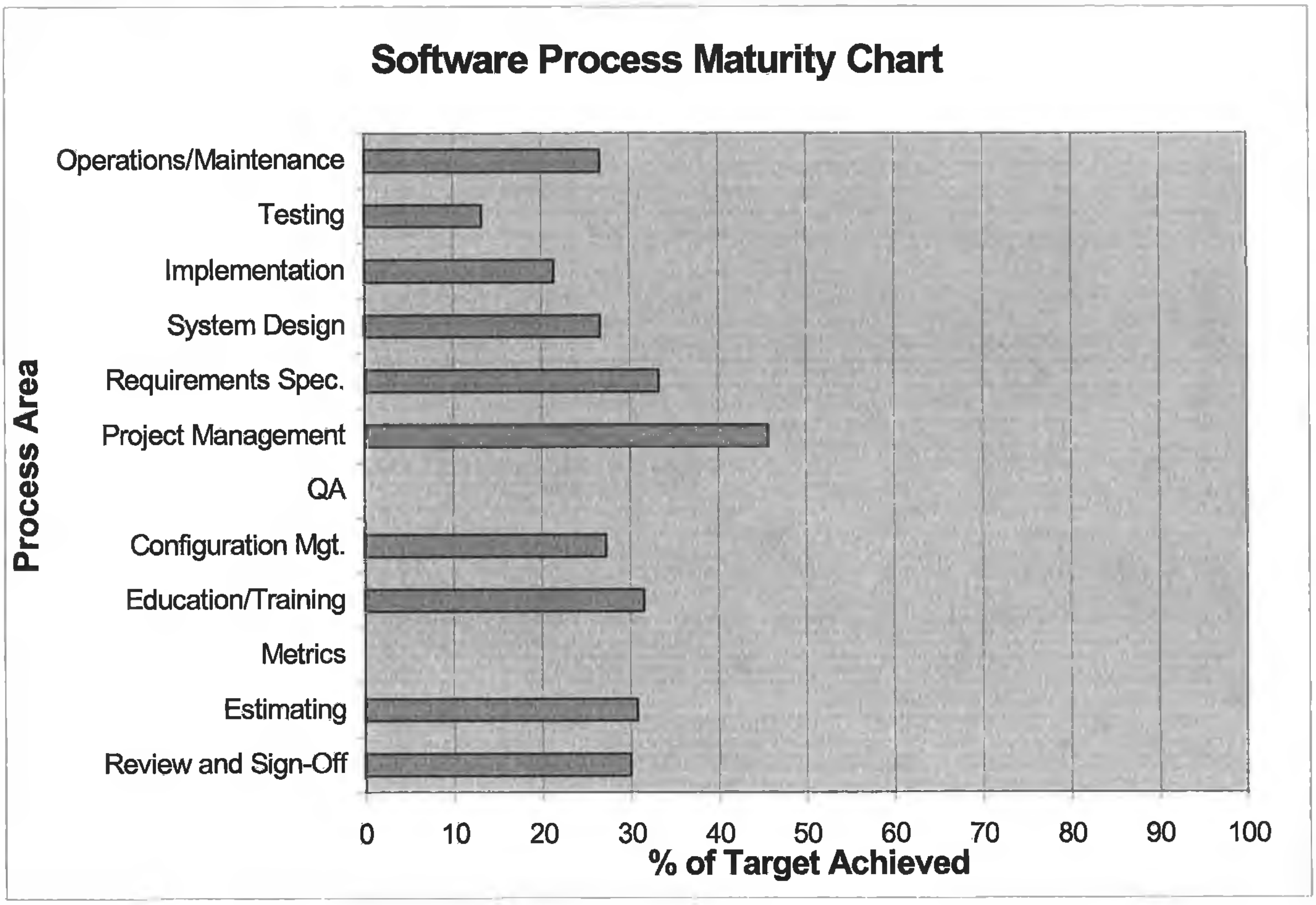

Figure 4.0 Company Process Maturity Chart

Below is a detailed record of the responses and any accompanying comments.

4.2.1 Project Review and Sign-Off % of Target Achieved 30

The questions related to whether senior management had a mechanism for reviewing a project's status and whether line managers sign off schedules and deliverables.

Comments suggested that

- ***a mechanism existed but was not well documented***
- ***the mechanism used was an informal discussion***
- ***deliverables are signed-off but schedules are not.***

4.2.2 Configuration Management % of Target Achieved 27

The questions in this section referred to whether the company had a configuration management function, whether this process was documented, how changes were made to software and how software releases were handled.

Comments included :

On whether mechanisms existed for controlling changes to the code

- Up to the individual programmer

- Executed by the Project Team

- Responsibility of management

On procedures for ensuring changes are reflected at every life-cycle stage

- Procedure exists on Unix platform but not on PC

- Up to the individual programmer

- Ad-Hoc.

4.2.3 Software Estimation % of Target Achieved 31

The questions in this section inquired whether formal procedures existed to estimate :

a) Software Cost

b) Software Size

c) Software Development Schedules.

Responses to:

a) ***Based on past experience and is documented***

Estimates for man-hours only and on an individual basis

An informal procedure exists

b) ***Up to the individual programmer***

c) ***Based on past experience and documented***

4.2.4 Metrics % of Target Achieved 0

This question asked whether any statistics, on software code and test errors are gathered.

From the responses it was clear that no metrics are collected, however one respondent commented that ***perhaps they may be collected on the current project.***

4.2.5 Education/Training % of Target Achieved 32

The questions in this section referred to whether:

a) Training activities are planned

b) Do staff receive the necessary training

c) Are sufficient resources provided for training

d) Are training programmes regularly reviewed.

Comments

a) ***Activities not formally planned***

Not very effective

Few courses for software developers

b) ***Different types of training are received***

There is no time available for training

c) ***There is a significant amount of 'on the job' training***

No training budget exists

Training is insufficient

d) ***Enough training is not being done***

Training programmes not formally reviewed

A training process is required.

4.2.6 Project Management % of Target Achieved 46

Respondents in this section were asked about

a) Project Planning

- Do procedures exist and are they followed
- Are project activities and deliverables documented and are estimates made in advance

b) Project Monitoring

- Do procedures exist and
- Are actual results compared with estimates and is action taken

Comments

a) ***Procedure exists but is not documented***

Procedure exists only for European Projects

Perhaps a procedure exists for European Projects

Procedures are followed for European Projects

The following of procedures is application dependent

Activities and deliverables are documented for European programmes

Programmers get a verbal indication and 1 page document on what system is supposed to do

b) ***Some checking of actual results against estimates is done by General Manager***

4.2.7 Software Quality Assurance % of Target Achieved 0

Questions in this section related to :

a) Whether the firm had a documented procedure for implementing SQA

b) Whether the organisation specified measurable quality goals

Comments

Company has used questionnaires to customers to determine quality of product

4.2.8 Requirements Specification % of Target Achieved 33

Questions asked here included

a) As requirements change are amendments made to plans, estimates and documentation

b) Does a written procedure exist for documenting requirements

c) Is the requirements process subject to SQA review

Comments

In relation to c) ***Discussions take place concerning progress***

4.2.9 System Design % of Target Achieved 27

Questions asked here included

a) As design changes are made are amendments made to plans, estimates and documentation

b) Does a written procedure exist for documenting design

c) Is the design process subject to SQA review

Comments

a) ***Up to the individual software developer***

There are no formal plans, estimates or documentation

Sometimes

c) ***Ad-Hoc***

4.2.10 Implementation % of Target Achieved 21

Questions asked here included

a) As design defects are discovered are amendments made to plans, estimates and documentation

b) Does a written procedure exist for documenting programs

c) Is the programming activity subject to SQA review

Comments

a) ***Up to the individual programmer - there are no formal plans, estimates or documentation***

b) ***Not as detailed as it might be***

c) ***Testing is used to check functionality***

4.2.11 Testing % of Target Achieved 13

Questions asked here included

a) As design defects are discovered are amendments made to plans, estimates and documentation

b) Does a written procedure exist for performing software testing

c) Is the testing activity subject to SQA review

Comments

a) ***Up to the individual programmer - there are no formal plans, estimates or documentation***

b) ***Incremental or ad-hoc testing is used***

4.2.12 Operations/Maintenance % of Target Achieved 27

Questions asked here included

a) As system defects are discovered are the necessary amendments made to documentation

b) Does a written procedure exist for documenting maintenance activity

c) Is the maintenance activity subject to SQA review

Comments

a) ***Amendments are made to programmer's documentation***

b) ***Log is kept by programmer***

A report will be filed

c) ***Details of maintenance activity will be kept by the programmer.***

4.3 Analysis of Findings

Cursory examination of the findings would suggest that the company has much to do to improve its software process. However, while the results highlight deficiencies in some areas they also reveal that significant progress has been achieved in others.

A lot of activity is carried out within the company in an informal way. This would be typical for a small company. Procedures and activities are not documented.

Information is communicated verbally and each employee understands where he/she stands in relation to the development activities. However without written procedures, misunderstandings do arise. Some respondents believed certain activities were documented. In answer to the same question others felt that activities were not documented but were understood.

On occasion respondents at managerial level felt the responsibility for certain actions lay with the individual programmer while in response to the same question developers felt that the responsibility lay with management. In some instances management were aware of certain tools which were available to assist the development process whereas developers were unaware of their existence.

From the responses, **Project Management** scored highest with the most formal approaches being adopted in this area. It is clear that procedures do exist for managing software projects however, they are not documented. Also, all development staff understand and agree their roles in advance of projects. For some developers, project plans are created to document the activities and deliverables for each project stage.

In the area of control and monitoring, estimates are made in advance of the project and action is taken if the actual figures deviate from the estimates. The drawback is that much of this is not documented or proceduralised.

There are two process areas where the company has yet to establish a presence :

Metrics and

Software Quality Assurance.

There was unanimity in the replies to the questions on Software Metrics and Quality Assurance. No attempt has been made, by the company, to collect any metrics during the software life-cycle. Also, no goals have been set for assessing product quality.

The only tool that has been used to date to determine the quality of the product is the elicitation of comments from customers. On occasion, customers have been issued with questionnaires to determine their satisfaction with the delivered system.

In the other areas such as Configuration Management, the Specification of Requirements and Education and Training the organisation has done some work and is edging towards a more mature process.

It is worth noting that the company's process becomes less mature as you move from the Requirements Specification through System Design and Implementation to Testing. Formal, documented standards and procedures are less apparent as one gets to the programming/testing level. Furthermore, because of the absence of any QA techniques, no guarantee can be given that the next system to be produced will be of the same quality as its predecessor. In the current situation much depends on the skill and dedication of the employees.

4.4 Summary

While a company remains small it is easier for management to retain some control over the quality of its products.

However, to retain a high-level of quality in software, as the organisation expands, a quality-oriented software process must be established. This will help to ensure that the systems which future customers receive will be of comparable standard to those which have already been delivered. Quality cannot be added to a software product at the testing or delivery stage. It must be engineered into the product as it is being developed. Building quality assurance into the software process will help guarantee the quality of the delivered system and ensure customers remain satisfied.

Whilst in the initial period this will be time-consuming the streamlining of the software process will enable the company to develop future systems more quickly, produce more accurate estimates of development time and have confidence in the quality of the final product.

Chapter 5 – Evaluation of the Development Environment within a Small Software Company

5.0 Introduction

Whilst the process maturity questionnaire gave a general overview of the company processes, it raised many conflicting views on how the software process operates and highlighted a lack of understanding or communication difficulties amongst employees. The purpose of this chapter is to attempt to overcome the confusion, that exists within the company, on roles, responsibilities and the operation of the software process. The approach taken, therefore, was to look in detail at how individual projects are tackled within the company, the forms and procedures used, and the development roles allocated to each employee. As part of this examination of the company, some recommendations were made on small measures that could be taken to improve the development process in the short term, prior to implementing more wide-ranging longer term improvements.

5.1 Process Assessment - Environment Evaluation

Prior to discussing the environment evaluation, some background company information is useful.

- Most of the company's products are based on Multimedia applications.
- All of the company's software output is bespoke. They have the customers in advance of development, and manufacture the product according to the user's specific needs.
- The user is involved at all stages of development. Within the multimedia area, feedback and input from the user are paramount if the system is to achieve its goals. Development is also an iterative procedure with changes and amendments being made based on user reaction.
- Projects may use different software development environments and hardware platforms.

The organisation in question is keen to adopt a quality approach to manufacturing software. The Centre for Software Engineering suggest that if a quality software system is to be established then it is essential that a particular life-cycle approach is adopted [CSE92]. With this in mind it was imperative to ascertain if the company developed products within any particular life-cycle model. This would be important as a future starting point for developing a standard software process within the company. Although the company believed that it used no particular development model, conversations with the development staff did suggest that there was an outline of a systematic method. However, this method was not 'visible' to the development staff and was not documented.

From my conversations with them it became more apparent that they had a way of doing things which had a degree of consistency attached. The approach could be diagrammatically represented as **Figure 5.0**. The design, construct and test elements, combine to form one iterative phase which is predicated upon user reaction to the delivered product. Ultimately, when user satisfaction is achieved the system is released. Although not always visible to the organisation this is the way they approach software development.

Significantly, the approach used by the company is closely aligned to the Evolutionary Development Model as outlined by Folkes and Stubenvoll [FOLK92].

The evolutionary approach centres around prototyping. The developed prototype is a working model of the system in its own right. The user can then evaluate the product and recommend changes. Based on the user's feedback the prototype may then be used to develop the next version of the system which the user can subsequently evaluate. The advantage of this technique is that users get to see a version of the system very quickly and are in a position to assess how closely their needs are being met.

Also the approach allows changes to the user's requirements to be incorporated at the earliest possible opportunity.

The user can also determine, very early in the process, if, say, certain system functionality is no longer needed. This feature of the method can save unnecessary development time.

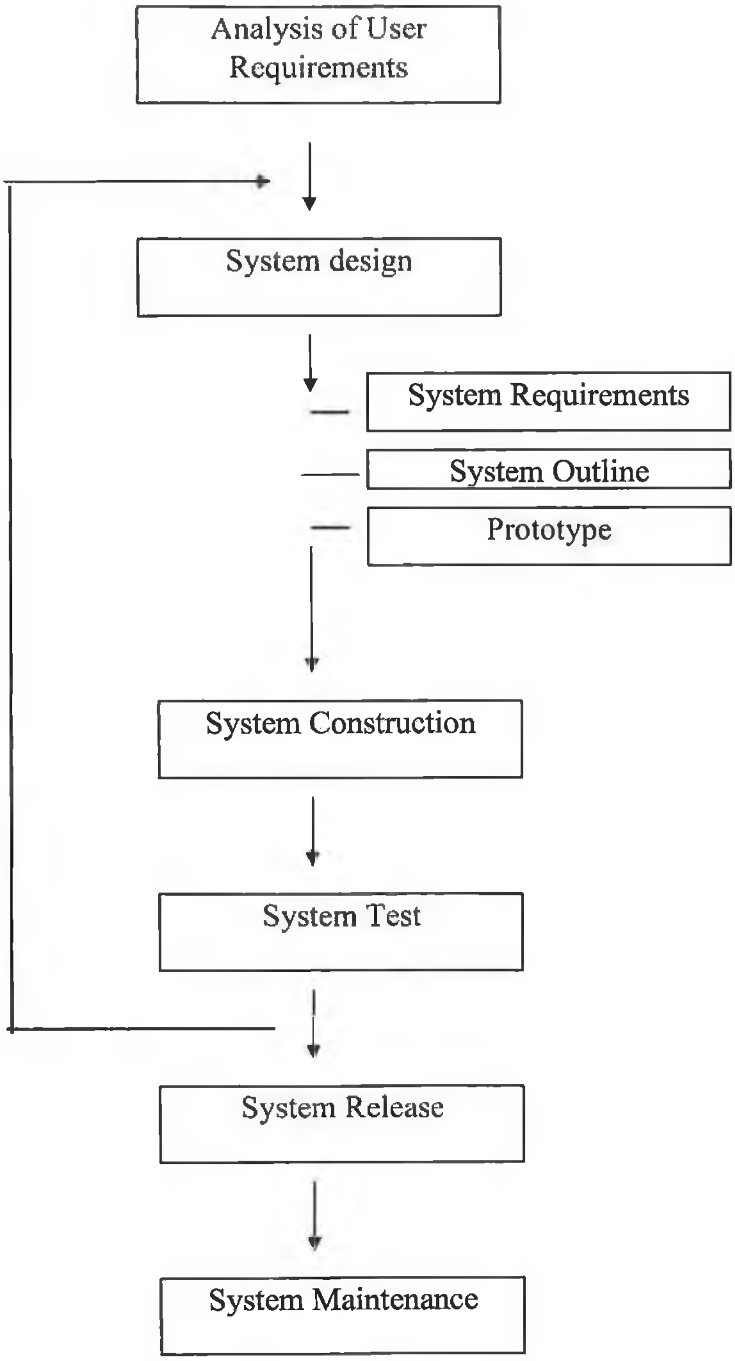

Figure 5.0 - Companv Development Paradigm

5.2 Development Strengths and Weaknesses within the Company

5.2.1 Development Strengths

Apart from the evidence provided by the questionnaire responses, the employees were also asked for their opinions on the company's strengths and weaknesses.

These are contained in **Table 5.0.**

Table 5.0 - Company Strengths and Weaknesses as perceived by its employees

Company Strengths	Company Weaknesses
Ability to get things done	Lack of Documentation
Good Working Environment	The Absence of Procedures/Standards
High Quality Work	No Measures of Quality
Use of up-to-date tools/technology	Insufficient Software Reuse

A wider look at the organisation's development process, introducing the questionnaire responses, shows that there are evident strengths within it. These lie particularly in the extent of user involvement in the development and the use of up-to-date tools and skilled personnel. Having extensive user involvement will help ensure that requirements are met and will improve the prospect of satisfaction with the delivered system. However, the development process, in its present form, does have some noticeable drawbacks and these will now be examined.

5.2.2 Development Weaknesses

Examination of the process, however, based on the questionnaire highlights more fundamental weaknesses not perceived by the employees. Apart from the absence of a defined software development process the company lacks formality and standards in other areas as outlined in **Table 5.1:**

Table 5.1 Company Development Process Weaknesses in the area of 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

1. **No Standard User Requirements Documentation** - Currently, the company works closely with the users in order to produce the User Requirements Specification, however, no standard document is produced at the end of this phase. Another problem is that no common format has been devised by the company for what should be included in/excluded from a User Requirements Specification. The absence of an agreed standard for this document will lead to confusion and misunderstanding on individual projects and prevent repeatability of success on subsequent projects.

2. **No Standard Design Documentation** - The company produces, initially, at this stage a Systems Requirements Document. Unfortunately, this also fails to conform to a standard layout with the result that the same problems associated with the User Requirements Document are evident. The organisation then produces an outline of the system which will be used in the prototyping stage. This again does not conform to any agreed standard. The document given to the developers is invariably composed of a few short pages of text. The absence of standardised documentation and approaches to design has many knock-on effects.

The system developers have to try and interpret a document written in purely natural language. The document they receive may bear little resemblance in format to documents previously received. For the developers to try and interpret this document unambiguously is an extremely difficult task.

As a result on many occasions the System Designer is required to sit alongside the programmers, often as they enter code, to ensure that the design requirements are met as closely and unambiguously as possible. Equally importantly, short and quickly prepared design documents, such as those issued to the programmer, cannot be used to any great extent during integration testing of the system and their overall contribution to ensuring the quality of the delivered product is at best negligible.

3. **No Programming Standards Exist** - The absence of programming standards will have a particular effect on system maintenance complicating error fixes and system enhancement.

4. **No Unit Test Plans are Produced** - The fact that no unit test plans are required from programmers makes the testing process less reliable. Formal test plans should at least ensure that the major system functionality is fully tested. However, unit test plans are also particularly important when program amendments are required after the product has been shipped. Their absence leaves the testing process less formalised and more subject to individual programmer diligence.

5. **No Formal Independent Testing of Modules** – Without independent testing, the emphasis is solely on programmers to ensure that programs are defect free. Also, without a defect detection and prevention system in place testing is used as the main error finding mechanism. Further, because no unit test plans are produced, the quality of the finished product will vary from project to project.

6. **No Formal Documentation of Errors** - Without this, measurement of the process cannot take place. Without measurement the company remains unaware of its strengths and weaknesses and this will render process improvement extremely difficult.

7. **No Formal Recording of Fault Reports and Change Requests from Users** Without logging these, the company is unaware of what problems their products are encountering in the field.

Also the number of errors found after shipping are not being distinguished from change requests or enhancements. Without this analysis the company cannot measure the quality of its products.

5.3 Comments on the Development Process

The study of the development process provides a snapshot of how the company manufactures its software products. There is no doubt that the company has achieved a considerable degree of success in its chosen marketplace. However, as can be seen from the analysis of the development environment this is due to the quality and technical skill of the staff and their hard work and endeavour rather than the application of standards and procedures and the creation of a quality process.

Discussions with the company have raised a number of important issues:

- They are keen to develop a quality process
- They use and are prepared to use up-to-date software tools and techniques
- They are prepared to address the deficiencies in documentation and standards
- They are prepared to make the necessary adjustments to the way they work in order to create the quality-oriented environment
- They are interested in collecting metrics and commence attempts to measure the process
- The commitment exists, from top management, to make the process work.

The above factors contribute greatly to the prospects of making a successful transition to a quality oriented software process.

5.4 Starting Point for an Improved Process

Having analysed the company and process approaches, which offer potential usage within small companies, this study will now examine how the transition can be made from the current company development environment to a new one based on quality.

With this in mind the DSDM approach to development in RAD projects could profitably be used here.

DSDM would provide the company with the quality oriented life-cycle it desires. The disciplines imposed by DSDM would introduce standardisation into an area where it is lacking. However, one of the advantages of DSDM is the flexibility it allows in development. The concentration is on fulfilling business requirements and so the company could easily meet a second desire of using up-to-date methods and tools as the RAD/Prototyping environment encourages this.

Furthermore DSDM's features do not impose unnecessary bureaucracy. This would be welcome in a small dynamic company such as this one. Its success involves getting things done and over-emphasis on documentation or procedures could stifle this attribute.

The company also expressed an interest in collecting metrics. Use of the PSP would provide an opportunity to do this. Metrics are a by-product of using the PSP allowing developers to track and measure their own process. Introducing metrics at an individual level will encourage their acceptance and allow for subsequent company-wide implementation. PSP will also be accompanied by standardisation and documentation. However, this will have to be introduced carefully as excessive recording and documentation could hinder development momentum. Nonetheless, if individual developers can adapt to a personal software process then the resultant work could have increased quality incorporated as a by-product.

Because of the noted deficiencies in the testing process the company have also expressed an interest in defect prevention (the hallmark of a good quality development process), and a more formalised testing process. A vague and ill-defined inspection process takes place currently. This mainly consists of an informal review of design documents and code. However, those charged with the task have not received any formal training in the inspection process.

The company is also interested in using software tools to assist the inspection process. Code analysers may be able to offer assistance in this area.

Developing a standardised software process will require the involvement and commitment of personnel, at all levels of the organisation, over a lengthy period of time.

There are, however, some simple measures, involving the standardisation of documentation, which could be implemented immediately and would assist in defining a software development process.

5.4.1 The Requirements Document

At present the requirements document agreed between the user and the analyst does not conform to a particular standard. The requirements document contains both the requirements definition and the requirements specification. The IEEE Standard 830 relates to Software Requirements Specifications and defines the areas to be included in such documents [IEEE84]. The requirements document should be composed of a series of chapters and, at a minimum, should include the following :

- Document title, Date of Production and Version Number
- Table of Contents
- Introduction (which may include background to and scope of the system)
- General Description of system
- Detailed Functional Requirements (This may include the detail of each of the system's functions; the input, processing and output involved; hardware and software requirements; database or file requirements; screens used etc.)
- Performance Requirements
- Glossary of Terms
- References (used in the document)
- Appendices (These may include, layouts of the screens as they occur during system operation, a list of the error messages within the system etc.).

If the company choose to follow a particular system development methodology then a format for the Requirements Document may be imposed.

The requirements document could then include diagrammatic sections encompassing such as, Data Flow Diagrams, System Diagrams, Entity Relationship Diagrams, Structure Charts, Decision Tables and Decision Trees and, if Object Oriented approaches were being used, Object Models, Object Interaction Diagrams etc.

5.4.2 The Test Plan

At present no test plans are produced either for unit testing or system testing.

While the detail of the design and build elements of the software life-cycle will be examined in greater later in the study, the company should endeavour immediately to devise a standard system test plan template. This should include, as a minimum :

- Document Title
- Project or System Title
- Test Plan Author
- Date of Creation of Test Plan
- Date (or range of dates) when Tests executed
- And for each Test
    - Test Number or Id
    - Test Description
    - Screen Sequence (if any) Expected
    - Expected Test Result
    - Actual Test Result
    - Fault Description (if any).

5.4.3 Using Fault Reporting

When the developed system goes live, even with a quality-oriented development process in place, after a period of time some software faults may emerge. However, given that errors will arise it is useful if a standard document, called, say, a Fault Report Form, is used to capture them. The company may also utilise this form when recording errors found in final testing, when, for example, tests are being executed in the public domain.

The form should contain the following:

- Document Title
- Project or System Title
- Fault Number or Id.
- Date Fault Occurred
- Description of Fault (including Screen Reference, Data Used etc.)
- Name of individual using system who encountered the fault
- Date of Fix
- Description of Fix
- Modules Amended
- Screens Amended
- Author of Fix.

Two files should be kept of the Fault Report Forms. One of these will contain outstanding, or unfixed errors, the other will contain forms where the errors have been fixed and a new software release has been issued to the user.

Having standard forms like these allow an archive of a system to be maintained. It may also be useful in defect analysis studies of the system and could be used in the collection of metrics.

Appendix A shows a proposed layout for such a form.

5.4.4 Change Request Form

When a system has been live for some months users will often request a change to the system.

This may be for a number of reasons:

- Because a bug has been encountered
- The company wants to incorporate extra functionality into the system
- New legislation has been introduced changing the way documents are produced
- A new management team has changed the way the company does business.

Any requests for changes to the system should then be recorded on a standard form.

This should include the following:

- Document Title
- Project or System Name
- User Name
- Date of Change Request
- Change Request Number or Id.
- Description of Change
- Reason For Change
- System Error/Enhancement Indicator
- Date of Change
- Description of Change
- Modules Amended/Added
- Screens Amended/Added
- Author of Change.

When a Change Request is subsequently executed, the standard test plan can be used for a regression test to ensure the change has not adversely affected any other system modules. Apart from the cross-referencing capabilities the archiving of Change Request Forms in the same manner as Fault Report Forms will contribute towards defect analysis and metrics collection. **Appendix B** shows a proposed layout for such a form.

5.5 Summary

The analysis of the company's operating methods has shown that software is developed, primarily in an ad-hoc fashion with limited standardisation and reproducibility. Significant changes are required to the software process if it to be standardised and capable of being both measured and continuously improved. The document standards outlined in **5.4.X** are not proposed as a solution to the problems inherent in the company's software process. They are, however, suggested as a starting point for the company to begin the process of drawing up standards and procedures. Moving from there to having a defined software process in place requires substantially greater changes.

The next section of the study focuses on these changes and recommends the use of DSDM and PSP and a means of achieving a defined, documented and measurable software process.

Chapter 6 – A Software Process for RAD

6.0 Introduction

So far the study has outlined the different types of process models that are available to software developers and has looked in detail at two particular models; DSDM for supporting RAD projects and PSP for improving the capability of individual developers. The study then examined a small software company with a view to determining the process used, the difficulties encountered in developing software and potential solutions to overcoming these difficulties. The purpose of this section is to propose how combining DSDM and PSP can be used as a solution to the software process problems experienced by the small company studied and other similar small companies. There are obviously certain issues to be examined when implementing new disciplines such as those contained in DSDM and PSP and these are discussed first. However, like any process, in order to determine the success of using these models, it is necessary to measure their effectiveness. With this in mind a range of metrics are suggested which can help measure this new develop environment and provide a roadmap for improvement.

Throughout this chapter, the phrase 'the company' refers to the company assessed and evaluated in Chapters 4 and 5.

6.1 Implementing a Software Process for RAD

Small companies looking to develop and use a standard software process need a starting point. DSDM and the PSP provide it. As the company works extensively on multimedia software and uses RAD tools and techniques, such as prototyping and iterative development, any process must facilitate these approaches. DSDM supplies this framework. Companies adopting this paradigm have given milestones and targets for which to aim. The Business Study and Feasibility Study elements of DSDM include the initial documentation standards which this company desires.

DSDM discusses what should be included in the Feasibility and Business studies and the accompanying exit criteria.

This is a critical starting point for the company as they are made aware of what documents are appropriate at each development stage and what should be included in those documents. The life-cycle framework also provides the company with a roadmap for development by clearly illustrating phase deliverables and resource requirements, thus allowing project monitoring and control to take place.

Working in the other direction, from the bottom-up, the PSP provides individual developers with the framework to develop and improve the own process. While DSDM provides the life-cycle around which the entire company can subscribe, the PSP allows individuals to improve their own performance. The PSP crucially imposes the collection of measures about the subscribing developer's performance. As has been stated previously in this study, one of the best ways to encourage process improvement is through the use of metrics. If individuals can witness the benefits of measurement then this could act as a precursor to a company-wide metrics programme. A company-wide programme would provide the springboard for significant process improvement. Used properly, metrics can supply the crucial data about an organisation's performance, highlight development strengths and weaknesses and lead to measurable improvement. However, meaningful metrics can only be collected if the development process is standardised and consistent which brings us back to adopting an appropriate life-cycle model.

A good approach to examining how to implement DSDM and the PSP into a small company is to use the Capability Maturity Model (CMM) as a benchmark for comparison. While CMM is primarily aimed at large software organisations, many of its 'key process areas' are equally applicable to small software development units.
The DSDM consortium state that introducing DSDM will help a company eliminate some of the ad-hoc practices associated with CMM level 1 and will address the key process areas specified by level 2. The PSP, on the other hand, covers key process areas from level 2 of the CMM, right up to the highest level, level 5.

Also, DSDM is an entire life-cycle approach. In the main, the PSP concentrates on the detailed design, coding and testing phase of projects.

Only when the PSP is implemented and fully understood by development staff could it be extended to other life-cycle areas. At that point the cyclic development process offered by PSP3 could be used in conjunction with DSDM.

The PSP is, essentially, a "metrics-driven" approach, with the measures collected during development being used to drive future changes and adjustments to both the process and software engineering practices.

DSDM is, on the other hand, a "requirements-driven" approach, with the emphasis on meeting user/business requirements in a short time-frame.

While there are major differences between DSDM and PSP and in how they can/should be used, there are areas for productive cross-fertilisation. The methodology, proposed in this study, can harness and integrate both paradigms.

6.2 Using DSDM within the Company

Certainly for a company, such as the one highlighted in this study, where there are no defined processes in place, DSDM is a good place to start. There are two questions which need to be answered:

- Is the organisation suited to the introduction of DSDM?
- Are the projects to be developed suitable for use with DSDM?

In answer to the first question, an organisation with no defined process and operating in an ad-hoc fashion certainly needs to begin by adopting a particular life-cycle.

The adoption of a life-cycle is the first stage in process definition. At the fulcrum of the DSDM approach is the use of prototyping. This brings with it its own demands - an iterative approach, developers capable of doing analysis, design, coding and testing, strong user interaction, a commitment to deliver a product quickly, and expertise in using development tools. Because of these strong demands, not every development environment is suited to the introduction of DSDM. However, the company featured in this study is a potentially good environment for the introduction of the method.

Furthermore, the type of applications developed by the company encourage a prototyping development approach. There are a number of important points which favour the use of DSDM within the company and these are documented in **Table 6.0**.

Table 6.0 - Factors which favour DSDM usage within company

Factors Favourable for 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

1. The company is small with multi-skilled staff. With current projects, many staff are carrying out analysis and design tasks in tandem with coding and testing. They develop bespoke systems with a particular emphasis on the multimedia area.

This, in turn, provides several further DSDM-friendly factors (user interaction, prototyping/iterative development, RAD tools).

2. Writing multimedia software requires a significant user involvement. One of the primary factors in multimedia applications is the user interface. A good user interface, requires significant user involvement during its development, a factor essential to DSDM.

3. With such applications, there will be a large component of iterative development. Prototypes will be written, demonstrated to the user and, based on user reaction and feedback, a new improved prototype will be developed. This process will continue until such time as the user requirements have been satisfied.

4. Another factor is the use of development tools. This company, by the nature of their business, are using the latest development tools. Prototyping requires tool support. A dynamic organisation, containing skilled staff, such as the one examined in this study, is more open to the introduction of new technology.

5. There is senior management commitment to process definition and improvement. Also, there is a drive towards quality as a factor in competitive advantage, particularly as some company projects involve European partners.
6. They wish to get their products on the market quickly as they again believe there is competitive advantage in this.

6.3 How Will using DSDM Benefit the Company?

DSDM will benefit the company in both process and product terms.
In section 2.10, the general benefits from using DSDM were documented. But there are some additional benefits for the type of company addressed in this study.
Initially, the company will be able to adopt a life-cycle model for its prototype-driven development. This will be the first step in process definition.
The ad-hoc system currently in place leads to lack of developer understanding, regular requests for clarification of program specifications from developers, rework, the absence of process measurement, no objective mechanism for assessment of process or product quality, the absence of scheduling, tracking and control mechanisms and the likelihood of being unable to repeat successful development approaches on successive projects.

Introducing a life-cycle model will provide a standardised software process from which to baseline development.
In 2.5 the DSDM principles were outlined. These fundamentals must be in place if the benefits detailed above are to be achieved. Significantly, several of these factors, including working with users, iterative and incremental development, the ability to reverse amendments during development, the baselining of requirements at a high-level and the co-operation of all relevant staff during development, are already in operation in the company. What is absent is the documented and defined process in which these factors can be optimally used. The use of DSDM, by the company, will provide this framework. Then with DSDM in place, the remaining fundamentals can be introduced.

In 2.5.1, the type of applications, suggested by the consortium, as suited to DSDM are listed. The first of these are applications where the user interface is of prime importance and where functionality is clearly visible. Multimedia applications, the primary application area of the company, unquestionably come into this category. The fact that such applications are so fundamental to the company makes the introduction of DSDM easier as there will be a greater choice of projects on which to pilot the method. Also because developers are more attuned to this iterative/prototyping approach, acceptance of this change will be more straightforward and the disciplines and rigours attached will seem more intuitive. All of these factors enhance DSDM's chances of acceptance and success within the company.

But what of the critical success factors listed in 2.5.3?

This is a more subjective area. Certainly the factors which are development specific, such as the use of prototyping and the adaptation of the latest tools and techniques are already in place within the company. What is less definite is the role of, and the access to, end-users, which is also required for success with DSDM.

The company operate as a software house. They write bespoke software for third parties. The concise role, which end users will play in any project, will depend on the organisation they represent. In companies where software is being developed for internal consumption then it may be easier to guarantee the required commitment of end users. Where third party bespoke development is involved, this is not so easy to ensure. If DSDM is to be used successfully, the company must convince the purchaser of the overriding importance of its having sufficient access to its users. If, however, the company does not have ready access to the purchaser's end users then the potential for using DSDM with the project is reduced, because in this case the ability to consult readily regarding requirements, prototypes and the user interface is diminished.

The company has, however, operated in such environments previously. Some of the work undertaken has involved European partners and system development has been shared. The company has foreseen the difficulties that such developments may encounter.

As a result it has successfully introduced up-to-date technologies to reduce risks. These technologies include ISDN which is installed within the company and available externally, videoconferencing facilities and extensive electronic mail and Internet connections. These innovations can ensure sufficient project momentum and help counter project risk associated with lack of direct access to users.

If the company has doubts about user involvement and support for the project, then the use of DSDM should be project specific. Because user involvement is paramount to success with the method, this should be evaluated initially to determine if DSDM can be used. If the requisite level of user involvement is not present then another approach should be adopted. This could take the form of prototypes being used in the early development stages to clarify user requirements and then, when this is achieved, the requirements could be frozen and a waterfall type approach used to complete the project.

6.4 Using PSP within the Company

As has already been illustrated in this study, the PSP is an attempt to scale down software engineering best practices to the individual level.

The potential benefits of introducing the PSP are substantial and its virtues have already been outlined in section 3.1. The analysis offered by Grady [GRAD93] is important in this context, where he discovered that changes to software development methods were easiest for employees to accept if backed up by metrics findings.

The results of the software process questionnaire discussed in Chapter 4 of this study, showed that there was a desire to introduce standardised ways of working. The employees believe that they produced good quality products and would be happy to have this ‘proven’. With this background the chances of the PSP gaining widespread acceptance and usage within the company are enhanced.

Earlier in the study we saw that the company had not implemented any form of metrics programme. Thus, it was impossible for them to ascertain their performance and to gain any objective assessment of their process and product quality.

The PSP enables metrics to be gathered as part of a defined process and as a by-product of normal working.

The factors which favour the introduction of PSP into the company are outlined in **Table 6.1**.

Table 6.1 - 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

For this company the PSP will be of particular use where waterfall-type approaches are used for project development, or as cited earlier, where ready access to users is not guaranteed. However, PSP disciplines can be introduced with DSDM projects and this will now be examined.

6.5 A New Development Environment

While the structure and approach of DSDM and the PSP disallow easy integration of the two methods, there are many opportunities for fruitful cross-fertilisation between them. DSDM was created to provide a defined life-cycle for RAD applications while the PSP was born out of the best practices inherent in the waterfall approach to software development.

However, taken individually neither approach offers the software process that is necessary for the company to achieve software of measurable quality in the long run.

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. Neither does it offer a life-cycle that can be used on a team-wide basis. Further, its focus is on the program design and coding elements of software development.

Taken together DSDM provides the life-cycle and framework which can be used by RAD teams and PSP offers the quality control mechanisms that are absent in DSDM. The PSP and DSDM can, therefore, be combined to form a quality software process for RAD, which will satisfy the software process requirements of the company.

6.5.1 A Quality Software Process for RAD – Combining DSDM and PSP3

PSP3 offers the closest match with the objectives and framework of DSDM.
PSP3 is the scaled-up version of the lower PSP versions and is suitable for larger projects involving more than one developer. PSP3 is a series of repeated PSP2.1 elements with each iteration increasing product functionality or system capability.
Figure 3.1 has already illustrated the PSP3 life-cycle. The Requirements and Planning stage, of PSP3, carries out similar functions to DSDM's Business Study; The High-Level Design phase resembles DSDM's Functional Model Iteration; and the Cyclic Development section is very much akin to 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. PSP3 is premised on the fact that the system requirements and the system design can be baselined and following this the system can then be developed using a sequence of increments. DSDM, by contrast, believes that the system requirements are to some extent fluid. While the Business Study will endeavour to baseline high-level requirements, it is assumed that lower-level requirements may be amended during development. Indeed if time pressures mean that the full system cannot be delivered within the allocated timeframe, then some functionality may be deferred to a future release. Implementing a development approach such as this within the company would be possible as a degree of iterative development takes place at present. Thus, what is required is that the existing iterative techniques are subsumed within a DSDM framework.

However, the PSP3 method does have some useful features which could be adopted within a DSDM project. PSP3 promotes some activities also argued by DSDM. 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. After each of the early cycle stages, detailed design, test plan development and coding a review is suggested. Closely adhering to this framework will ensure that 'clean' versions of software are used as input to subsequent system increments. This factor should be built into DSDM developments. The company could incorporate this relatively easily.

The software process questionnaire results has shown that reviews are already used within the company albeit in a haphazard and unstructured way. As the commitment is already present to use reviews some extra training of appropriate employees will improve the usage of review techniques.

The questionnaire responses also indicate that in this small company developers have great responsibility and roles are not as rigid as perhaps in a larger company.

Developers, therefore, have experience of not just coding but analysis and design also. This will be invaluable for adopting DSDM as speed of development is maintained if the development team are skilled and empowered to make decisions.

While the fact that only programmers tested their own modules was seen as a weakness in the process, based on questionnaire responses, it is inevitable that this will happen in DSDM projects. This is because momentum must be maintained at each life-cycle phase. More importantly, it reduces the overall test burden ensuring that all of the testing is not left until coding has been completed. Also it ensures that defect-free modules can be passed from one stage to the next.

This requires the commitment of experienced personnel. That commitment is present within this company. The High-Level Design phase of PSP3 includes, as an activity, the identifying of the product's natural divisions. It suggests that a good benchmark is for each cycle to produce between 100 and 300 lines of new and changed source code. Having historical productivity figures for developers will enable estimates to be made of cycle development time, or in DSDM, how much code/functionality can be delivered within, say, a given timebox.

If , for example, historical productivity is 20 LOC/Hour and an estimate for certain system functionality is 3000 LOC then it can be estimated that, on a utilisation factor of 6 hours per day, a developer will take 25 days or 5 working weeks to produce the desired functionality. In some projects it may be that LOC is not an appropriate measure. In systems which are heavily GUI dependent then modules, screens or sessions which are more user-visible may be more appropriate. Within DSDM, using the activities proposed by the PSP's High-Level Design, will allow the natural boundaries for timeboxes and prototypes to be established.

These figures, however, are not a natural by-product of using DSDM.

Indeed the Consortium does not specify how the necessary metrics are to be collected. PSP can be used in this regard. By getting individual developers to collect these metrics it makes the subsequent implementation of a company-wide programme easier. Without these metrics the estimates needed by DSDM for timeboxes, prototyping and overall schedules will not be easily produced and may be inaccurate.

In the way outlined above, the PSP has made allowance for iterative approaches to be used. It is important to note, however, that each PSP3 cycle is essentially a PSP2.1 process that produces part of the final product.

As such, it is fundamental to examine the lower levels of the PSP and establish ways in which they can be integrated with DSDM.

6.5.2 A Quality Software Process for RAD - Using Proxies

DSDM, as has been illustrated, has five life-cycle stages. At PSP levels 0 to 2, a process is defined for the activities of detailed design, coding and testing. These activities will be carried out in DSDM in the two iterative prototyping stages, the Functional Model Iteration and the Design and Build Iteration.

The objective of the Functional Model Iteration is to demonstrate the required system functionality and to highlight the essential non-functional requirements. This is done through the production of a prototype.

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.
In order to assist the estimates of how much functionality can be delivered within a timebox, the consortium suggest the use of function point analysis.

While the PSP supports FPA as an estimating technique, its own favoured approach is through the use of Lines of Code (LOC) as the base measurement unit. The PSP expands on these size/complexity techniques through its use of ***Proxies***. Proxies, which act as code substitutes from which program size can be determined, were discussed in 3.4.1. From historical data you can determine how many LOC or FPs can be developed within the timebox. Further, as historical data amasses and estimating skills improve, additional proxies can be developed for other programming languages. Use of proxies in this way will improve estimating techniques in DSDM projects and help ensure that business functionality is met.

At present estimates are made by the company in advance of projects. These estimates are primarily based on the experience of the project manager and are not backed up by historical figures. In the absence of these figures 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. If they are aware of the relationship between software size and development time then they can bid for contracts more confidently. The PSP will allow them to collect the necessary figures. They could then relate estimates of software size with developer productivity to assess with confidence and accuracy how long a project would take.
These productivity measures could also be used within the proposed quality software process for RAD to state what functionality could be produced within a given timebox. Without PSP and the accompanying measures this would not be possible.

6.5.3 A Quality Software Process for RAD - 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; an admission that testing is not the most effective way of detecting and eliminating errors. The consortium also advocate independent testing of software (i.e. the software is tested by someone other than the author), and recommend that the user carries out this activity.

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 inspection, 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 may mean the passing back and forth of software products between developer and user, notwithstanding the possibility of substantial rework. While it is laudable and desirable for users to be involved in acceptance testing, theirs should be a further quality assurance element.

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'. Whilst there are potential benefits to using these products, their success will depend on how skilled the developers are in using them and the language dependency of the particular tool.

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.

Also in this company, with no history of formal inspections, the sole use of software tools to carry out this work would be undesirable. This is because the employees will not gain sufficient understanding of the importance of inspections and will come to rely on the tools to do all the work. The tools should be used as support only when a manual inspection process has been in place, used extensively, understood and properly applied.

6.5.4 A Quality Software Process for RAD – A Quality Plan

In 6.5.3, it has been shown how the defect detection techniques advocated for use with the PSP can be used within the proposed quality software process for RAD. The DSDM consortium also address other quality issues which are of interest and which can benefit from PSP methods. In the PSP, the quality emphasis is on defect management and this provides the foundation on which a comprehensive quality strategy can be built. DSDM takes a broader view of quality. 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.

Obviously in early projects, the plan itself will evolve and will require evaluation and subsequent improvement. Small companies will not normally have such a plan and, indeed, the company under examination in this study has yet to develop one. Therefore, the creation of a Quality Plan will be a primary task.

Adopting the proposed new software quality process will provide an approach to development which contains the procedures for controlling quality in a RAD environment. However, the techniques of process definition outlined in the PSP can assist in the creation of a Quality Plan.

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 which again ensure quality coverage of the process. Including these in a quality plan introduces the standards and control elements demanded by DSDM. Each time, for example, a prototype is being developed the software engineers can refer to the relevant scripts in the quality plan. This will ensure that they are conforming to the quality criteria decreed by the organisation and will result in greater consistency of results and the increased prospects of repeating successes on future projects.

In the company we have studied, the introduction of these documents and techniques as part of the proposed quality software process will provide two elements which are currently absent; a documented and defined development method and quality control measures.

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 could be established for each language used by the organisation. Furthermore, as the company introduces new development environments, these too could be added to the quality plan. The PROBE estimating script, introduced in PSP1, could also be included in the quality plan. This exists to assist the estimation process and will be very useful in assessing product deliverables within DSDM timeboxes.

At present the company attempts to make estimates prior to project commencement. These are essentially based on experience and guesswork. The adaptation of PROBE, which with enthusiastic developers could be successfully achieved, would produce greater estimating accuracy and supply engineering disciplines to their software process. Another PSP document which would be part of the quality plan is the Process Improvement Proposal (PIP) already referred to in section 3.3.1 as 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. Also, a mechanism should be established to ensure that suggestions made on PIPs are evaluated and implemented, where appropriate.

Having a quality plan enables you to gain control of your process. Once you have done this you can then examine how to improve it.

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 our quality software process for RAD. The prerequisite for using these measures in the new process is the introduction of the metrics detailed in the next section.

6.5.5 A Quality Software Process for RAD - Metrics

At present, the DSDM consortium is not recommending any specific approach towards collecting metrics. However, they do recommend recording the following by timebox:

- The business functions delivered
- The effort expended
- The elapsed time
- The size of the prototypes in such as, function points, lines of code etc..

They also suggest that these records should be kept for each prototype developed.

While these are undoubtedly useful measures, no counts are being kept of any errors introduced during the timebox or prototype development. 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 many 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, it may also be difficult to ascertain what development activity caused the error i.e. analysis, design, coding etc. which, in turn, will make it difficult to determine where the weakness in the process lies.

Thirdly, 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.

Fourthly, there will be no picture available of how effective the interim testing is i.e. testing of timebox elements, prototypes etc.

Fifthly, although the consortium state that ‘in a DSDM project, task-based time recording is an unnecessary overhead’, without doing so process weaknesses and deficiencies will not be highlighted.

DSDM also 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.

It is proposed that errors be categorised by type in DSDM projects. Five categories are suggested, however, the Defect Type Standard Table (**Table 6.2**), taken from the PSP, is more comprehensive and could be used in the new process.

Some additions or adjustments could be made to the table to take account of GUI environments. Defects relating to control positioning on forms or incorrect property values should be documented. Developers will likely create their own standard in these cases which can be language or environment dependent.

Table 6.2 Defect Type Standard - Sample from PSP

Purpose		To facilitate cause analysis and defect prevention	
Note		The types are grouped into ten general categories. • If the detailed category does not apply, use the general category. • The % column lists an example type distribution.	
No.	Name	Description	%
10	Documentation	comments, messages, manuals	1.1
20	Syntax	general syntax problems	0.8
21	Typos	spelling, punctuation	32.1
22	Instruction Formats	general format problem	5.0
23	Begin-end	did not properly delimit operation	0
30	Packaging	change management etc.	1.6
40	Assignment	general assignment problem	0
41	Naming	declaration, duplicates	12.6
....	....	....	....
....	....	....	....

Within the new process, defects should be recorded by timebox and by prototype, using the PSP approaches. Analysis of defects will show process deficiencies. Improvement in the process can then be achieved by tackling these areas where deficiencies exist.

As the new process will be prototype-driven, then the timebox should be the basic unit of time in which the measurements are collected. Measurements can also be collected by prototype phase.

If metrics are to be collected accurately then it is important that they are based on the new and changed code developed in each iteration. Because of the iterative nature of RAD developments, a Base Program will be used as input to each iteration.

As such, productivity is not centred around the total lines of code or total function points resulting from that iteration but the new and changed code/function points produced in that iteration. Therefore, it is the new and changed code/function points that are essential to the validity of the metrics. 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 important to note that it is desirable not to introduce all these metrics at once into a project. Some companies may have specific weaknesses, such as, large numbers of development defects, which they may wish to tackle initially.

It is only through using and collecting metrics that companies or individuals can decide which are the most appropriate and cost-effective to collect. As competency in metrics collection improves, then the range of measures gathered can be widened.

6.5.5.1 Defect Metrics

The defects will be measured relative to code size. Size can be measured either per new and changed function point (NFP) or per thousand new and changed lines of code (KLOC). Defects can then be measured as follows :

$$\text{Total Defects/Size (Timebox)} = \frac{\text{Total defects per Timebox}}{\text{Size per Timebox}}$$

The effectiveness of the testing process can be assessed through :

$$\text{Test Defects/Size (Timebox)} = \frac{\text{Test defects per Timebox}}{\text{Size per Timebox}}$$

The above measures can also be used with prototypes :

$$\text{Total Defects/Size (Prototype)} = \frac{\text{Total defects per Prototype}}{\text{Size per Prototype}}$$

ALSO

$$\text{Test Defects/Size (Prototype)} = \frac{\text{Test defects per Prototype}}{\text{Size per Prototype}}$$

Ideally a defect database should be maintained containing information about the defects injected and removed during the development. The database should contain, such as, Program Number, Defect Number, Defect Type (using, say, the Defect Type Standard illustrated in Table 6.2), Inject Phase, Remove Phase and Fix Time.

The above fields are proposed for use with the PSP. As a result, they are based on the design, coding and testing areas. These fields, however, could be adjusted for use with the iterative development approach that is central to DSDM. The Program Number field could be changed to Prototype Identifier, to reflect the DSDM approach. 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 and can illustrate not only the type of defects being injected but within what type of object, function and timebox. This can subsequently be related to the type of functionality being delivered in the given timebox and, therefore, provide a more complete picture of when the defects are injected, the nature of the modules/functions associated with the introduction of certain defect types and what sort of time/deliverable pressures results in what type of defects.

The fix times will throw further light on the process's defect removal capability. When compared against the defect types and the phases introduced it can show where deficiencies lie. These figures can be used to assess the DSDM process. Because of time constraints an organisation may wish to use automated tool support for code inspections and testing.

The figures contained in the database can be of great assistance in evaluating both manual and automated procedures, if they are in place, and prove a measure of the capability of the organisation.

At present, in the company, no metrics are collected. Questionnaire respondents indicated that the introduction of quality control measures would be welcome and would be embraced by employees.

Defect metrics are probably the best and easiest measures to introduce to this company first. Whereas developers may have more difficulty adapting to and understanding the implications of size and schedule metrics, they can instantly identify with defect metrics. Using DSDM and the PSP make it easy to collect defect metrics and employees can see their own performance at first hand. However, for such a programme to be successful in this company, it is essential that the goodwill of the developers is not abused by using these defect metrics in such as performance appraisal. Management have an opportunity in this instance to demonstrate their goodwill in relation to the use of these measures.

Handled properly, it will allow other process measures to be introduced and the co-operation and support of the development staff to be gained.

6.5.5.2 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:

$$\text{Productivity} = \frac{\text{Total Additions}}{\text{(Development Time Unit)}}$$

Productivity measures should also be calculated by prototype :

$$\text{Productivity (Prototype)} = \frac{\text{Total Additions (for prototype)}}{\text{Total Development Time Units (for prototype)}}$$

The productivity measures could also be calculated per timebox :

$$\text{Productivity (Timebox)} = \frac{\text{Total Additions (for Timebox)}}{\text{Total Development Time Units (for Timebox)}}$$

In waterfall developments, coding is done at a specific stage, post software design and prior to testing. In RAD environments, some coding will be done at the early stages to clarify requirements, and subsequently there will be an iterative cycle of designing/coding/testing throughout the development with the ever-present prospect of rework. Using the LOC/Hour measure for total development will not necessarily give a true productivity reflection in this environment. It is essential to measure productivity by timebox and by prototype. In doing so, a better indication of bottlenecks and timebox/prototype accuracy is recorded. For example, a process may be weak at the functional model iteration prototype phase. Developers may be experiencing difficulty progressing from the Business Study, and coding the prioritised functions from the agreed architectures and designs.

This difficulty will be hidden if LOC/Hour or FP/Day is based on Total Development Time. If this calculation is taken at the prototyping phases then such weaknesses will be highlighted. Similarly, calculating LOC/Hour or FP/Day at the Timebox level will expose any deficiencies which may exist at the different development phases.

In this company the productivity metrics will need to be treated carefully. They will complement the size metrics. 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. Also in many stages of development large portions of time may be spent in user consultation.

This again may reduce crude LOC productivity figures but is essential if business functionality is to be met and user satisfaction achieved. Similarly, if there is significant code reuse then again productivity figures based on LOC may be underestimated. The company must devise appropriate measures of productivity. These should be different for projects which use different development environments. For example, multimedia projects which involve substantial user interface design and user involvement should be treated differently than other company projects involving lots of file handling employing 3GL code.

These measures should be agreed in advance between developers and management. The key factor is consistency. The same measurement techniques must be used for each development, based on project type, if reliable metrics are to be collected and acted upon.

6.5.5.3 Size Metrics

Size can be counted at all stages of development and can be based on Lines of Code (LOC) or Function Points(FP) delivered. What would be useful in DSDM, and to ensure minimum disruption to development, is automated support for counting FP or LOC. These figures will be necessary for both the productivity and the defect measures to be calculated and, therefore, need to be collected not only for overall development, but also by prototype and by timebox.

The figures can be used to determine the estimation errors in code size.

Size estimation errors can be calculated as follows:

$$\text{Error\%} = 100 * \frac{(\text{Actual Size - Estimated Size})}{\text{Estimated Size}}$$

Size can be counted as Lines of Code (LOC) or delivered Function Points (FP).

While the estimation error can be calculated for the total delivered code, it would be useful in DSDM to measure it by prototype and by timebox.

By prototype

$$\text{Error\% (per prototype)} = \frac{100 * (\text{Actual Size of Prototype } - \text{ Estimated Size of Prototype})}{\text{Estimated Size of prototype}}$$

By timebox

$$\text{Error\% (per timebox)} = \frac{100 * (\text{Actual Size of Timebox } - \text{ Estimated Size of Timebox})}{\text{Estimated Size of Timebox}}$$

In DSDM, a corollary of this is the actual functionality delivered within a given timebox. Each organisation may have its own method of assessing functionality, whether it be functions, objects, modules, screens, files etc. In our company the size metrics have significant importance for other metrics. Because of the current widespread use of RAD tools, the fact that the company is edging towards software reuse and the prevalence of multimedia systems, Line of Code counting is not necessarily the best way to estimate size. In some company projects, where multimedia is not used and 3GLs are the standard, LOC then have relevance.

However, this company should look at Function Point counting in some detail. This would more closely reflect the way the development environment operates and would give credit towards time spent on screen design, developing reusable modules, interfacing with users etc.. The size metrics outlined above can operate with either the Function Point or the LOC approach.

6.5.5.4 Time/Schedule Metrics

For future estimates of development time or deliverable capability per timebox it is necessary to collect time measures. This facility is present throughout the PSP where the time spent on the various phases is documented.

These time measures are then used to determine how much time was spent on development in each of the particular phases. It feeds into the productivity measures for determining LOC/Hour etc. If the productivity measures are to be accurate then the analysis of time spent must also be recorded per prototype and per timebox.

For scheduling, the PSP techniques can be used to plan timeboxes and check whether the promised deliverables have been completed within the timebox. The Schedule Planning Template and the Task Planning Template from the PSP can be adapted for use with DSDM timeboxes and prototypes. The Earned Value concept referenced in 3.8 can be used to measure the success of achieving delivery targets. All of the collected measures can be used as future inputs to timebox delivery estimation and scheduling.

Time estimation errors can be calculated as follows:

$$\text{Error\%} = \frac{100 * (\text{Actual Time - Estimated Time})}{\text{Estimated Time}}$$

Time can be counted in the most appropriate units e.g. hours, days, weeks etc..

While the estimation error can be calculated for the total development time, it would be useful in DSDM to measure it by prototype.

The necessary calculations would be:

Error% (per Prototype) =

$$\frac{100 * (\text{Actual time to develop Prototype - Estimated time to develop Prototype})}{(\text{Estimated time to develop Prototype})}$$

There may also be agreement between users and developers to apply priority weightings or percentages to specific items of functionality. To check its capability, the organisation should then measure the delivered functionality at the end of each timebox, prototype and on project completion. This is crucial as, in order to stay within time schedules, some lower-prioritised functionality may have been jettisoned. 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 can also be calculated by prototype and timebox:

$$\text{DR (Prototype A)} = 100 * \frac{\text{(Delivered Functionality in Prototype A)}}{\text{(Planned Functionality in Prototype A)}}$$

$$\text{DR (Timebox A)} = 100 * \frac{\text{(Delivered Functionality in Timebox A)}}{\text{(Planned Functionality in Timebox A)}}$$

Importantly, the measure could be used at the end of any iterative phase or timebox to determine the ratio of delivered functionality to date:

$$\text{DR (Milestone A)} = 100 * \frac{\text{(Delivered Functionality to Milestone A)}}{\text{(Planned Functionality to Milestone A)}}$$

Gathering all of the above figures will assist in refining the estimation process. Also, in this way, a picture of the development process can be established and the figures used to feed into future estimates. Time and Schedule metrics are particularly important for small software companies where, typically, only a small number of projects are under development at any one time. The importance of accurate measures for our company in this regard cannot be overstated. Also, having reliable time and schedule figures allows the company to make more accurate bids for software development projects. This would be particularly important in fixed price contracts where financial loss can occur as schedules slip and deadlines are subsequently extended.

6.5.5.5 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 from a phase over the total number of defects removed and remaining:

$$\text{Yield (step n)} = \frac{100 * (\text{defects removed in step n})}{(\text{defects removed in + escaping from step n})}$$

Review Yields refer to the percentage of defects in the design or code at the time of the review that were found by the review. This can only be categorically determined when the reviewed code has been tested and subsequently used. Review Yields could be used to check both the quality of the review procedure and any automated tools used to carry out inspections or reviews.

$$\text{Review Yield} = \frac{100 * (\text{Defects Found By Review })}{(\text{Defects Found by Review + Escaping from the Review})}$$

This approach could be further refined to assess prototype and timebox yields.
For example, a prototype yield could be:

$$\text{Yield (Prototype)} = \frac{100 * (\text{Defects found in Prototype})}{(\text{Defects Found in Prototype + Escaping from the Prototype})}$$

And a timebox yield would be:

$$\text{Yield (Timebox)} = \frac{100 * (\text{Defects found in Timebox})}{(\text{Defects Found in Timebox + Escaping from the Timebox})}$$

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.

$$\text{Defects/Hour} = 60 * \frac{\text{(Defects Removed in Phase)}}{\text{(Minutes in that phase)}}$$

$$\text{DRL (Design Review)} = \frac{\text{Defects/Hour (Design Review)}}{\text{Defects/Hour (Unit Test)}}$$

These approaches could again be profitably employed in DSDM by measuring defect removal rates in prototypes and timeboxes. This will be more effective, as the results will be provided more quickly because of iterative development and the regularity of testing. The defect removal rates will provide essential information in making comparisons between early prototypes and later prototypes and between early and later life-cycle timeboxes. Again the review measures would be useful in assessing the proficiency of any automated tool used in inspections/reviews.

The measures could be calculated as, for example:

$$\begin{array}{l}\text{DRL (Code Review) =}\\ \quad\text{for Prototype A)}\end{array} \quad \frac{\text{Defects/Hour (Code Review for Prototype A)}}{\text{Defects/Hour (Unit Test for Prototype A)}}$$

OR

$$\begin{array}{l}\text{DRL (Design Review) =}\\ \quad\text{for Timebox A)}\end{array} \quad \frac{\text{Defects/Hour (Design Review for Timebox A)}}{\text{Defects/Hour (Unit Test for Timebox A)}}$$

Review yields illustrate the quality of your development process. Taken in conjunction with test yields they show where defects are being uncovered. High review yields are preferable to high test yields as the former indicate that defects are being found earlier in the process and are thus cheaper to correct. High test yields indicate escapes from the other quality control mechanisms and are more costly to fix so the figures provided by these measures carry great significance. DRLs show the defect removal capability of your process. You are looking for high defect removal leverage in the review phases over the testing phases.

Again this is an indication of the success of the early quality control mechanisms as this study has already shown that reviews are a faster and more cost effective way of removing defects than testing.

6.5.5.6 Cost of Quality (COQ)

There are a number of Cost of Quality (COQ) measures associated with the PSP which address the time spent in review as a ratio of time spent in test.
The Failure Cost of Quality is the percentage of total development time spent testing and compiling while the Appraisal Cost of Quality is the percentage of total development time spent in design and code review. These measures could also be used in DSDM. The PSP Appraisal COQ is designed to illustrate that time spent reviewing means less time spent testing. The Appraisal to Failure Ratio (A/FR) is the ratio of Reviewing to Compiling/Testing. The higher this ratio, the more time spent reviewing and the less testing. However, the use of, for example, Static and Dynamic Code Analysers will reduce the amount of time spent reviewing. In this instance the Appraisal COQ may be quite low. Therefore caution is advised in making comparisons of these figures. As such, different Cost of Quality measures are required in RAD environments. The Delivery Ratio has already been proposed as a potential quality measure for DSDM. This together with the Yield metrics already discussed will provide a better indication of the Cost of Quality than those associated with the PSP.

The extended measures will be useful in this company as they start to give a true indication of the capability of the development process. When the process has been measured in this way, it can be examined and improvements identified. The extended measures should be introduced after the basic measures have successfully been implemented and assessed.

6.5.6 Maintenance

Projects developed in RAD environments have, in the past, been criticised for being unmaintainable. Companies, however, use RAD because they wish to develop systems quickly. Delivering systems quickly, though, may mean that maintainability factors are sidelined during development. The quality software process for RAD proposed in this study will improve the maintainability of projects through improving their quality. Because of this, maintenance efforts should primarily be directed at product enhancements.

This does not mean that post-delivery defects should not be collected. The PSP allows for post-delivery defects to be included in the defect counts. A useful measure of the quality of the development process would then be the Live to Development Defect Ratio (LDDR):

$$\text{LDDR} = 100 * \frac{\text{Defects found in live environment}}{\text{Defects found during development}}$$

In order for this to be truly meaningful the defects would have to be categorised as to whether they were actual defects uncovered in the software (Corrective Maintenance), requests for software changes due to changed business requirements (Adaptive Maintenance) or requests for system enhancements. Proposed formats for collecting the details of change requests and fault reports, as suggested for use by the company examined in this study, are included in Appendices A and B.

Another useful measure of the maintainability of the product would be a measure of the time spent in corrective maintenance. This would be the actual time spent fixing defects after development. The best measure here would be the average defect fix time post-delivery versus the average fix time prior to delivery. The PSP forms proposed for use with DSDM will collect the fix times for defects both during and after development.

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:

$$\text{DFTR (Post-delivery / Pre-delivery)} = \frac{\text{Average Defect Fix Time (Post-Delivery)}}{\text{Average Defect Fix Time (Pre-Delivery)}}$$

This measure could also be extended to illustrate the time spent fixing defects within development phases e.g. fix time per defect in testing versus fix time per defect in say design or code review. The DFTR and the previous measure the LDDR will be of particular relevance in RAD projects as it will help illustrate the 'hacking component' of the project and the phases of the development where the defined software process was not properly followed.

The company in question has, at present, no indication of the number of errors occurring in the field. It also does not distinguish between errors reported and change requests generated. The only measures it uses currently are qualitative ones.
Customer questionnaires have been produced to ascertain satisfaction with the finished product. The responses, however, have not been quantitatively assessed. Use of the maintenance metrics proposed would provide important information as to where developers were spending their time, for example on new systems or the corrective or adaptive maintenance of existing ones.

Information such as this would be very useful to the company for resource allocation and future staff projections. It would also highlight the weight of the maintenance burden. In speaking to the employees it was clear that developers spend a lot of time answering customer queries and fixing bugs, based on customer requests, on an ad-hoc basis. No records are kept of the amount of time developers spend doing this type of work. In many instances the developers, deciding that the fix is a simple one, make it on the spot. Also there are often lengthy phone calls dealing with customer queries. Again this particular activity is not documented. As such getting an overall picture of how developers are spending their time is impossible. This is where the maintenance metrics have a role.

Even if the company were to spend one month just tracking the time spent in these areas there could be a real payback in terms of raising awareness of the true costs involved.

What activity like this does show however, is the fact that skilled developers make design decisions and intuitive guesses of how long fixes and rework is going to take. This ability could be developed and harnessed within a quality software process for RAD.

6.5.7 Software Reuse

Both DSDM and PSP state that reusing existing software components will result in productivity and quality improvements.

In section 2.2 the belief, that reuse will increase the speed of development and reduce the time-to-market, was shown [HENR95]. Also, there is a fear that some RAD methods because of their quest for fast delivery ignore the engineering and maintainability issue with software having to be rewritten if it is to be reused.

It is here that there are sharp differences between DSDM and the PSP. While DSDM believes there are a number of benefits to reuse it does not believe that DSDM projects should bear the costs of enabling that reuse. The DSDM consortium states that though RAD developers should use, where possible, existing software components there is a risk that 'the DSDM team may start building components for future reuse, incurring development costs not related to the DSDM project objectives.

This should be actively discouraged'. Therefore to enable reuse in this type of RAD environment, the costs of reuse could be charged to a different cost heading.
In contrast, the PSP has an objective of developing 'the personal process disciplines needed to produce reuse-quality software'. Indeed within the PSP forms, areas are reserved for documenting both the amount of existing code reused and the code developed which is destined for future reuse.

Both disciplines are correct. There is no doubt that there are significant productivity benefits to reusing software. Corporations can profit from having libraries of reusable components accessible to every new project. 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. This is undoubtedly an organisation cost. If developers are to reuse these parts then they must satisfy themselves that the components meet the functionality requirements and are of sufficient quality to be reused. There are also the costs associated with the extra time required to develop reusable components.

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. Again there are costs with this development, however, these may not be time-constrained.
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. The individual developer, however, can still maintain his own component library.
Whatever policy is adopted concerning software reuse will, of course, have an effect on system maintenance and maintainability.

This company has shown a willingness to develop reusable components. Without a defined software process in place such activity could prove wasteful and unworkable.

It is only through a defined, standardised software process that proper software reuse can exist. This environment is provided though the proposed quality software process for RAD.

Developers must be encouraged to reuse software. In this company they have tried to do this without the proper support mechanisms. These mechanisms, libraries, dictionaries and references can only exist and be properly managed within a defined process. As such, to gain maximum benefit from reusability and to prevent wasted effort, a defined process should first be put in place before software reusability becomes fully achievable. With a controlled, measured process in existence the introduction of proper corporate reuse libraries can be created. This will allow maximum benefit to derive from an area that has huge potential.

6.6 Summary

The Personal Software Process (PSP) and the Dynamic Systems Development Method (DSDM) are based on two separate paradigms.

The PSP has emerged from the waterfall approach to software development, which is a sequential technique for software development, whereas DSDM has emerged from the iterative approach to software production. Both approaches have their advantages and disadvantages. What is at stake is the ability to harness the beneficial elements of each method into one process. This chapter has focused on this particular opportunity and has proposed a new development environment using a quality software process for RAD.

In the study we are dealing with a company which though effective is operating with no defined software process. In the absence of a software process, the company has no way of measuring its performance. This study has focused on how two methods, suitable to such a company could be used to provide a defined and measurable software process. This chapter has illustrated how the two methods could be introduced into the company. Ways of using and integrating the two methods have been discussed. It has been shown how the metrics proposed by the PSP, for measuring defect rates, time and schedule estimates, productivity rates and quality measures could be adapted for use in a RAD environment.

Also, new metrics, for determining functionality delivery capability and for assessing live defect rates have been detailed.

The benefits of software inspection and review as defect detection techniques, and how they can be introduced within an iterative development approach were also highlighted. Also discussed were the implications for the testing of prototypes and the usefulness of testing as a quality assurance measure.

The chapter has further outlined how maintenance could be handled in the environment, comprising PSP and DSDM, and discussed the factors affecting software reuse where the two paradigms offer competing agendas.

Within the chapter it has been explained how, if all of the measures outlined were implemented within the company, as part of a quality software process for RAD, it is anticipated that not only would the company then have a defined and measurable process, but that this would produce lower defect rates, higher productivity, reduced schedules and quality products.

In addition, the company would now have a basis for continuing software process improvement.

Chapter 7 – Independent Review

7.0 Introduction

The purpose of this section was to have the proposed process model, combining DSDM and PSP, validated by software professionals experienced in the field.

These professionals have expertise in working with small companies and in using various life-cycle models and approaches and, as such, are well-qualified to comment on whether or not the proposed approach is suitable for its chosen environment.

7.1 Review Panel

The review panel comprised 3 experienced practitioners in the areas of RAD and Software Process Improvement: Mr. Patrick O'Beirne (Systems Consultant, Systems Modelling Ltd.); Mr. Shay Curtin, Product Director, Kindle Banking Systems); and Mr Gerard McCloskey (Lecturer in Computer Science, Letterkenny RTC and formerly Project Manager, Information Technology Centre, Letterkenny).

Their comments on the proposed methodology and the author's response are detailed below:

7.2 Independent Review Comments

Mr. Patrick O'Beirne:

The study has shown an interesting liaison of DSDM and PSP.

Firstly, re. the PSP, it is important to use the self-convincing nature of the PSP course. People used to sales pitches don't believe documentation but they believe their own experience.

Company Assessment [Chapter 5]

On the company assessment - it's nice to see the way an objective assessment can reveal "hidden" flaws.

A Quality Software Process for RAD [Chapter 6]

The DSDM emphasis on the user leads nicely into the PSP emphasis on the design aspect, where PSP assumes the requirements are 'obtained'.

Regarding metrics, in the PSP they are personal. The public interface needs to be agreed. This happens once any work is passed onto someone else. In 'prima donna' development, this may not happen.

I like the points about DSDM being top-down, requirements-driven and the PSP being bottom-up, metrics-driven. I also think there is good matching between the company and the project to DSDM.

Section 6.5 'A New Development Environment' *is of particular interest to me. I think that in RAD areas LOC is not really appropriate and perhaps user-visible things like modules or sessions are more suitable.*

I agree with the statement that 'it's easier to calculate how much can be done in a certain time..[pp.87]' as that is how practitioners do it. It's possible because 'how much functionality' is a much fuzzier measure than 'how much time'.

I think perhaps reviews could be regarded as tests within DSDM.

I couldn't agree more with the paragraph challenging the assertion that 'confidence is derived from finding errors [pp.88]'. It exposes the confusion between 'confidence in the testing process' and 'confidence in the development process'.

You could perhaps adjust the defect type standard table [pp.93] to include references to GUI environments e.g. '10 minutes lining up controls on the form' will make people ask is there a better way of doing things. Answer - 'hold down the shift key while you click ...'

With so many metrics it's worth specifying the point about climbing Everest one stop at a time. In other words talk about incremental improvement.

Companies must ensure that defect metrics are not used in performance appraisal. It's worth considering how this issue can be raised, discussed and resolved in a trusted manner.

I notice you make reference to the difficulty of productivity/total time hiding phase inefficiencies [pp.97]. They serve different purposes: one is for estimation and the other is for process improvement. The key word is not just 'consistency' but 'fitness for purpose' which I see you also subsequently refer to.

Be careful of using the phrase 'figures can be misleading' in the section on Appraisal COQ and Failure COQ [pp.104]. Any figure can be used to mislead. It can only be validly used in the context to which it applies. NOT collecting data just because one process is different from another is scarcely useful. Flagging the inadvisability of comparing such figures is important. The Appraisal COQ may be low, but be careful of using words like 'true reflection'. It's true as far as it goes.

I liked the suggestions on maintenance and the metrics associated with assessing maintenance effort [Section 6.5.6, pp.105]. Regarding initial improvements to the company's process, if the company did nothing else (in terms of process improvement) for one month but track that time, there could be a real payback in raising awareness of true costs.

Regarding the section on software reuse [section 6.5.7, pp.107], is DSDM really anti-reuse? Maybe they just mean that it should be charged to a different cost heading than the current project. It's a good sink of time and money for developer gold-plating. I wholeheartedly agree with the section on ensuring that a proper reuse infrastructure exists if reuse is to be instituted [pp.108]. You can only reuse code (or modules, or screens) if you can find them, and the IDE makes it as effortless as pasting in a built-in language element.

<u>Author's comment</u>

Mr. O'Beirne raises many important issues in this section. Firstly on the issue of Lines of Code (LOC) not being an appropriate standard for measurement in RAD, I agree with him on the whole. Because of the prevalence of the user interface and the saving in development time (and code) offered by GUIs LOC may not be the most appropriate measure.

The reservation I have, is that in companies that have not previously introduced metrics, LOC is an easier measure to understand and introduce and the concept and implementation of function points requires greater skill and training which may not be available to such a company.

Mr. O'Beirne's reference to the defect type standard not including GUI references is valid. The PSP has essentially derived from waterfall approaches and third generation languages.

Adjustments could be made to the standard to include defects on user interface design, control design etc.. Indeed I have made some reference to this in the section in the study on defect metrics.

His point about not using metrics in performance appraisal is an important one. If management do use them in such a way then the process is discredited and developers will subsequently not 'buy-into' the process and could become deliberately mislead regarding defects generated, time spent on project phases etc.

The PSP insists that the metrics generated are personal. Metrics generated company-wide could be discussed at QA fora within the company. Calculated at a project level, particularly within a resource pool environment, they can be effective as figures will not then be individually attributable. Indeed if developers can be encouraged to 'buy-into' the process then they may wish to remedy their own weaknesses. This can be successfully achieved in a non-threatening environment and should be encouraged.

I accept his criticism regarding comparison of figures for Appraisal Cost of Quality and Failure Cost of Quality and have adjusted the text accordingly.

Finally regarding my comments on DSDM being anti-reuse, I was suggesting that there may be a conflict between RAD and reuse at the development level. The emphasis in RAD is on producing a product that is fit for purpose in a specified timeframe. Designing for future reuse may hinder that development as more effort will be required in component design to ensure that it is reusable. Indeed in DSDM they talk about project components being re-engineered for reuse following project delivery. A trade-off may be required between RAD and reuse if both approaches are to be successful.

Mr. Shay Curtin:

I think your theory of marrying PSP and DSDM is excellent, well researched and very well worked through. I've always had difficulty relating 100% of the theory to practical software development processes and all of this difficulty revolves round the "soft" factors. Conditions must be ideal for theory and practice to be indistinguishable and any observations I have on your study are related to this.

DSDM is a sound methodology and you have covered the issue of access to users and the bespoke nature of the projects involved. Ultimately the majority of software development companies become software product companies where the emphasis is on off-the-shelf or near off-the-shelf products. In this scenario, DSDM is still useful as a specification development methodology, where once specified and agreed the production follows the more classical waterfall process.

PSP, like any software process, requires the buy-in of everyone involved - in the case of traditional projects, the measurement is of the whole of the effort involved and this "averages-out" the skill differential and the effect of the "soft" factors (i.e. Company/Management style, environment, incentives, motivation, ...).

PSP relies heavily on individual self-measurement and the "soft" factors can influence the accuracy of recorded statistics.

Author's comment

Mr. Curtin's references to the 'soft' factors involved in software development are very pertinent. Ultimately, it comes down to whether the developers 'buy-into' the process or not. If not, then the recorded figures will be at best unreliable and at worst deliberately distorted. Mr. Curtin mentions the factors that have an effect on this; management style, culture, motivation, incentives etc.. If developer support is not present for any process or approach then improvement is unlikely. Mr. Curtin's suggestion of using DSDM as a specification approach is interesting. The difficulty with this is guaranteeing the subsequent speed of development when then reverting to more traditional approaches to complete the project. Also user involvement, in the project, may be systematically reduced as traditional approaches take precedence. This in turn could effect the delivered system's chances of satisfying the user's requirements and being 'fit for purpose'.

Mr. G. McCloskey:

A number of the points made in the 'Company Assessment' section [Chapter5, pp.63] I won't wholly agree with but I suppose someone looking from the outside can take a much more clinical and 'fairer view than people working within the company everyday. The approaches of DSDM and PSP mentioned in ***section 6.1*** *[pp.76], in my opinion, have a number of disadvantages.*

My main concerns about using the approach would be when would the prototyping end and the reliance on the end-users, currently guiding system development correctly, begin. While developing multimedia applications for public use, constant interaction with small user groups does not guarantee success.

Also reading through your study I was unsure as to what exactly a RAD was.

Is it dependent on the development environment, the number of lines of code, or the language? I suppose my question is if I am developing over 10K lines of C++ using a visual IDE; is this RAD? Also complexity should be a factor here.

The approach does not put forward anyway of using the benefits of classes and inheritance supplied by C++ and JAVA.

The iterations suggested during each stage of development would greatly increase development time. In some projects where profit margins are small development would be occurring at a loss.

For PSP my main concern would be staff evaluating themselves and the difficulties of monitoring their progress based on these results. The approach of documenting errors and actions is good but if the company is working on numerous small projects simultaneously the overhead may be outweighed especially when developers do not log all their errors.

The fact that the overall approach is more of a conglomeration of existing development and testing methods it maybe cheaper/quicker/better for companies to readjust their current approach to areas in DSDM where they believe they are weak.

Author's comment

Firstly, end-user involvement is there to improve the chances of success of the project. I agree there is no guarantee of success but results have shown that user involvement does increase the chances of the users being satisfied with the finished product and of its meeting their requirements [LUQI91, GORD95, JACQ94, KERR94].

On the issue of RAD there is indeed confusion in the industry as to what constitutes RAD. RAD is essentially a paradigm in which projects are delivered in a short space of time. This time is divided into timeboxes in which incremental elements of the system are delivered. Following these basic rules RAD can be independent of platform and development environment. In practice, however, companies are using advanced development tools to produce systems within short timeframes. DSDM provides a quality framework in which this can be achieved and also supplies a 'suitability filter' to gauge whether projects are suitable for RAD or not.

I take issue with the fact that the method does not supply a way of utilising the classes supplied by C++ and Java. A section of the study is devoted to reuse and argues in favour of reuse as being a provider of quality and enabling RAD. Certainly at the Business Study and Functional Model Iteration phases of DSDM reusable components should be identified. As stated in the study this will only be successful if a properly supported environment for reuse exists.

On the suggestion of the iterations increasing development time, a **maximum** of three iterations are recommended by the consortium. If the developers can meet the requirements in one iteration then great!

On the difficulties of monitoring staff progress using PSP, I disagree with this point. PSP should be used by developers to monitor their own progress. It should not be used by management in the same way.

Regarding the overhead of developers recording figures if working on several projects, it is conceded that there will be an overhead involved in recording figures. However, the developers will be recording their own figures for estimation, defects etc. across projects. They can if they wish record it by project but again these figures are for their own consumption.

Finally Mr. McCloskey suggests that organisations could adjust their own process to fit in with areas of DSDM where they think they are weak. I have reservations about this.

Firstly it assumes that the organisation has a defined process. In the organisation studied as part of this study no such process existed and introducing a process was important.
Secondly, if an organisation does not collect any metrics it cannot say for certain in what areas it is weak. Only by collecting metrics will strengths and deficiencies be highlighted.
Thirdly, as the DSDM consortium states, using this approach will only move companies to level 2 of the CMM. To progress to a more mature development organisation, companies will be required to refine their process a lot more if further improvement is to be achieved. This study has provided a road map for how this can be achieved in certain small companies.

7.3 Summary

The independent review undertaken as part of this study has revealed a number of important issues and, I believe, strengths of the method proposed.
All three contributors made reference to the importance of the developers believing in the process and recording the figures. I agree that this is paramount. However, if figures are to be collected then this must be done within a well-defined process; if the process cannot be defined then it cannot be controlled and measured.
Overall I believe that the method provides a route to a more mature process for small companies who have yet to define a process. The proposed methodology can also be adopted by companies who develop projects in a RAD environment and wish to start measuring their process in order to improve it.
I believe the approach outlined in this study provides a framework for achieving A Quality Software Process for Rapid Application Development and I therefore welcome the positive responses and the support given by the reviewers for the proposed method.

Chapter 8 - Conclusions

8.0 Introduction

This study is concerned with improving the software development capability of small software organisations. The study proposes using a process model, which combines the top-down, RAD-oriented framework of DSDM with the bottom-up, metrics driven approach of PSP. The findings have been validated by three software professionals to determine the usability of the proposed model in small software companies.

The purpose of this chapter is to document the results of the research and to recommend areas suitable for further study.

8.1 Detailed Analysis

The objective of this study was to examine the software process in a small software development environment and to propose ways in which this could be improved.

Two new methods, DSDM and PSP, were examined to determine their suitability and application in small software organisations.

A company was then chosen, an overview questionnaire completed and interviews with company staff undertaken to determine the methods used by the company to develop software, the tools employed and the standards and procedures followed.

What emerged was a small dynamic company, composed of skilled staff using undocumented and ad-hoc approaches.

The prevalence of ad-hoc development approaches in such circumstances was clear and as a result the organisation could be said to be immature. This, nevertheless, would not be untypical for a small software company. It was against this background, that the analysis was undertaken.

Two development paradigms, DSDM and PSP, were examined to identify what benefits they could offer and whether they were suitable for a small company.

The study revealed that implementation of DSDM would provide a road map for development and a means of achieving Capability Maturity Model (CMM) level 2.

While DSDM provided many advantages, the company also needed to begin measuring its performance in terms of estimating, scheduling and particularly the quality of both its products and process. These performance measures are supported at the individual level by the Personal Software Process (PSP).

The next aspect of the study was to outline how DSDM and PSP could be used together as process improvement mechanisms within the company.

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, PSP does not preclude the use of tool support in these areas. Trained, experienced personnel using appropriate tools, such as static and dynamic analysers, can ensure that project momentum is maintained.

Also PSP3, through its support for cyclical development, supplies the synergy that is required to operate with DSDM. PSP3 supports unit and integration testing at each cycle and this is congruent with the iterations produced in DSDM.

Other benefits also accrue from using PSP and DSDM together. The proxy-based estimating technique of PSP will help with defining timebox elements in DSDM.

PSP also supplies appropriate metric-collection techniques which can be used in DSDM projects. Also some additional metrics were proposed which take account of the particular requirements of RAD projects.

DSDM projects require a quality plan. The documents and scripts associated with PSP provide a major input into this plan and together with the metrics-collection approaches allow you to plan for quality, monitor it and check that quality targets have been achieved.

This study, therefore, proposes that both the PSP and DSDM could be profitably used together, in small companies, as part of a quality software process for RAD.

8.2 Recommendations

8.2.1 Data Recording Assistance

The rigours of form filling and data recording, associated with the PSP, may cause problems in the new software process environment.

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 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 study, carried out in a Canadian company, has employed a different approach for the collection of PSP time recording measures [SHOS96]. In CAE developers were encouraged to maintain detailed logs of their time on randomly selected days only.

This would reduce substantially the potential development delays generated by the rigid data collection features of the PSP.

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 recording would be advantageous.

Any tool which can assist with time recording would also prove effective. Should such a tool not be available then the provision of an on-screen clock or 'stopwatch' would be an interim measure. The PSP provides a spreadsheet which generates all the statistical data and charts necessary to assess progress. Unfortunately, the user must enter the data manually into the spreadsheet first. Assistance in automatically transferring this data from development to the spreadsheet could be profitable.

8.2.2 Inspection/Review Assistance

The PSP places great emphasis on code reviewing and inspection as methods of finding defects early in the process. 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.

8.2.3 DSDM Software Process Improvement Measures

DSDM has been designed to provide a life-cycle model for RAD. However, within DSDM there is no account taken of software process improvement measures.

Within the DSDM manual it states that DSDM can be viewed as a process. Although the process is particularly well-defined, few measures are included for improving the process. While version 3 of DSDM may include process improvement measures, PSP measures can be used for process measurement and improvement in the intervening period.

8.3 Future Research

Having examined the background of both DSDM and the PSP and proposed how they could be utilised within a small software development company, future research could concentrate on a number of areas.

1. Implementing the measures within a company.

The adoption of these approaches by a company will allow the determination of their efficacy within the suggested environment.

The results emanating from such a study will allow for the refinement and improvement of the paradigms and how best they can effectively be cross-fertilised.

The time measures will be of particular interest as they will illustrate by how much the recording mechanisms inherent within the PSP will affect development speed and schedules within DSDM.

It is proposed in early projects that, where appropriate, only DSDM is used in situations where the project criteria correspond with those discussed in 6.2. Similarly, in early projects which follow the criteria in 6.4, only the PSP should be used. Employing the paradigms in this way allows developers to become familiar with the standards imposed by each approach. Then when sufficient experience has been achieved, and early feedback from their use evaluated, the migration path for amalgamating them, where appropriate, can be determined.

2. New ways to measure quality - how can customer satisfaction be measured?

At present the PSP proposes measures for both the product and the process. Most of those measures concentrate on, for example, the number of defects within the product and adherence to schedules. One of the only measures used after the product has been shipped is the count of the number of post-delivery defects. Future research could concentrate on objectively quantifying customer satisfaction with the delivered system, through the use of, perhaps, usability metrics or other such measures. Pre-development agreements including clauses where the customer 'must be 100% satisfied' with the product could then be more objectively managed.

3. CASE environments to support metrics collection and RAD

Future research in this area could also examine the prospect of developing integrated CASE tools to support the entire RAD development process.

Also effort could be put into verification and validation tools which could perhaps, take in as input a system design written in a form of standard pseudocode. Also a tool which permitted the entry of user-defined design checklists against which the formalised design could be compared would be advantageous.

4. *Defect Prevention vs. Defect Removal, Causal Analysis, Statistical Measures*

Future work may also wish to explore the area of defect prevention and causal analysis.

Causal and statistical analysis could provide interesting information on how defects are introduced in RAD and what are the similarities/differences between those and the ones produced in waterfall developments.

Statistical analysis of historical data will allow companies to predict the number of defects within a product after shipment. Better still, by analysing the cause of defects, process deficiencies, such as insufficient developer experience, inadequate training, poor process documentation etc. can be highlighted.

Indeed it is only in this way by measuring and examining their activities that companies can increase their software process maturity.

9. References

[ACKE89] Ackerman, A. Frank, Buchwald, Lynne S.& Lewski, Frank H. 'Software Inspections: An Effective Verification Process', IEEE Software, May 1989, 31-36.

[BALZ83] Balzer, R, Cheatham, T.E. and Green, C. 'Software Technology in the 1990s: Using a New Paradigm' IEEE Computer, November 1983, 39-45.

[BENI56] Benington, H. D. 'Production of Large Computer Programs' Proceedings Symposium on Advanced Programming Methods for Digital Computers, Office of Naval Research, USA, 1956, 15-27 (see also Proc. 9th. Int. conf on Software Engineering, 1987 (IEEE Computer Society Press 299-310.)).

[BOEH76] Boehm, Barry W. 'Software Engineering' IEEE Trans on Computers, December 1976, Vol. c-25, No. 12., 1226-1241.

[BOEH88] Boehm, Barry W. 'A Spiral Model of Software Development and Enhancement' IEEE Computer, May 1988, Vol. 21, No. 5, 61-72.

[BROO87] Brooks, F. P. 'No Silver Bullet: Essence and Accidents of Software Engineering' IEEE Computer, April 1987, 10-20.

[BURG95] Burgess, Angela 'Mad About or Mad At Measurement' IEEE Software, January 1995, 115-117.

[CARD95] Card, David 'The RAD Fad: Is Timing Really Everything?', IEEE Software, September 1995, 19-22.

[CARM93] Carmel, Erran 'How Quality Fits Into Package Development', IEEE Software, September 1993, 85-86.

[CARM95] Carmel, Erran 'Does RAD Live Up to the Hype?', IEEE Software, September 1995, 25-26.

[CSE92] The National Centre for Software Engineering, *Framework for Success: A Guide to Quality in Software Development and Support*, National Centre for Software Engineering, 1992.

[CURT93] Curtis, B. & Paulk, M. 'Creating a software process improvement program' Information and Software Technology, June/July 1993, Vol. 35, No. 6/7, 381-386.

[CUSU89] Cusumano, Michael 'The Software Factory: An Historical Interpretation', IEEE Software, March 1989, 23-30.

[DAVI94] Davis, Alan M. 'Fifteen Principles of Software Engineering', IEEE Software, November 1994, 94-101.

[DORL93] Dorling, A. 'SPICE: software process improvement and capability dEtermination' Information and Software Technology, June/July 1993, Vol. 36, No. 6/7, 404-406.

[DROM96] Dromey, R. Geoff 'Cornering the Chimera', IEEE Software, January 1996, 33-43.

[DSDM95] *Dynamic Systems Development Method*, Version 2.0, DSDM Consortium, November 1995.

[FAGA76] Fagan, M.E. 'Design and Code Inspections to Reduce Errors in Program Development', IBM Systems Journal, 1976, Vol. 15, No. 3, 182-211.

[FAGA86] Fagan, M.E. 'Advances in Software Inspections', IEEE Transactions on Software Engineering', July 1986, Vol. SE-12, No. 7, 744-751.

[FOLK92] Folkes, Susan and Stubenvoll, Sue *Accelerated Systems Development*, The BCS Practitioner Series, Prentice Hall, 1992.

[FREW86] Frewin, G.D. & Hatton, B.J. 'Quality Management - Procedures and Practices, Software Engineering Journal, January 1986, Vol.1, No.1, 29-38.

[GORD95] Gordon, V. Scott & Bieman, James M. 'Rapid Prototyping: Lessons Learned, IEEE Software, January 1995, 85-94.

[GRAD93] Grady, Robert B. 'Practical Results from Measuring Software Quality', Communications of the ACM, Vol. 36, No. 11, November 1993, 62-68.

[HAAS94] Haase, Volkmar, Messnarz, Richard, Koch, Gunter, Kugler, Hans J. & Decrinis, Paul 'Bootstrap: Fine-tuning Process Assessment' IEEE Software, July 1994, 25-35.

[HANN95] Hanna, Mary 'Farewell to Waterfalls?', Software Magazine, May 1995, Vol. 15, 38-46.

[HARD95] Hardgrave, Bill C. 'When to Prototype: Decision Variables Used in Industry', Information and Software Technology, 1995, Vol. 37, No. 2, 113-118.

[HENR95] Henry, Emmanuel & Faller, Benoit 'Large-Scale Industrial Reuse to Reduce Cost and Cycle Time', IEEE Software, September 1995, 47-53.

[HUMP88] Humphrey, Watts S. 'Characterising the Software Process: A Maturity Framework' IEEE Software, March 1988, 73-79.

[HUMP95] Humphrey, Watts S. *A Discipline for Software Engineering*, Addison Wesley, 1995.

[IEEE84] IEEE Guide to Software Requirements Specifications, IEEE Standard 830, 1984.

[JACQ94] Jacques, Trevor 'RAD Takes Developers Across the Waterfall', Computing Canada, January 1994, Vol. 20, 22.

[JOCH95] Joch, Alan & Sharp, Oliver 'How Software Doesn't Work: Nine Ways to Make Your Code More Reliable', Byte, December 1995, 49-60.

[KERR94] Kerr, James & Hunter, Richard *Inside RAD - How to build fully functional computer systems in 90 days or less*, McGraw Hill, 1994.

[KITC96] Kitchenham, Barbara & Pfleeger, Shari Lawrence 'Software Quality: The Elusive Target', IEEE Software, January 1996, 12-21.

[LIND93] Lindstrom, David R. 'Five Ways to Destroy a Development Project', IEEE Software, September 1993, 55-65.

[LINT95] Linthicum, David S. 'The End of Programming', Byte, August 1995, 69-72.

[LUQI91] Luqi & Royce, Winston 'Status Report: Computer-Aided Prototyping', IEEE Software, November 1991, 77-81.

[McCR92] McCracken, D.D. and Jackson, M.A. 'Life-Cycle Concepts Considered Harmful' ACM Software Engineering Notes, April 1992, 29-32.

[McLE93] McLeod Jr., Raymond *Managing Information Systems: A Study of Computer-Based Information Systems*, 5th Ed., Macmillan, 1993.

[MART91] Martin, James *Rapid Application Development*, Macmillan, 1991.

[MONT95] Montgomery, John 'Make Quality Job 1', Byte, December 1995, 54.

[MYER93] Myers, Ware 'At Forum on Quality, Emphasis is on Testing', IEEE Software, September 1993, 94-96.

[OLSE95] Olsen, Neil C. 'Survival of the Fastest: Improving Service Velocity', IEEE Software, September 1995, 28-38.

[PAUL93] Paulk, Mark C., Curtis, Bill, Chrissis, Mary Beth & Weber, Charles 'Capability Maturity Model, Version 1.1', IEEE Software, July 1993, 18-27.

[RAFI95] Rafii, Farshad & Perkins, Sam 'Internationalising Software with Concurrent Engineering', IEEE Software, 1995, 39-46.

[REIL95] Reilly, John P. 'Does RAD Live Up to the Hype?', IEEE Software, September 1995, 24-26.

[ROYC70] Royce, W. W. 'Managing the Development of Large Software Systems: Concepts and Techniques' Proceedings IEEE, Wescon 1970 (see also Proc. 9th. Int. conf. on Software Engineering, 1987 (IEEE Computer Society Press 328-338.)).

[SHOS96] Shostak, Barry 'Adapting the Personal Software Process to Industry', Software Engineering Technical Council Newsletter, Winter 1996, Vol. 14, No. 2, 10-12.

[SIMO95] Simon, Ray 'Software Development needs Modern Approach', National Underwriter (Property & Casualty/Risk and Benefits Management), September 1995, Vol. 99, No. 5, 20.

[SOMM92] Sommerville, Ian *Software Engineering*, 4th Ed., Addison Wesley, 1992.

[SPMQ94] 'Software Process Maturity Questionnaire', Software Engineering Institute, Carnegie Mellon University, 1994.

[THOM93] Thompson, K., & McParland, P. 'Software Process Maturity (SPM) and the Information Systems Developer' Information and Software Technology, June/July 1993, Vol. 35, No. 6/7, 331-338.

[WELL93] Weller, Edward F. 'Lessons from Three Years of Inspection Data' IEEE Software, September 1993, 38-45.

[WOHL94] WohlWend, Harvey & Rosenbaum, Susan 'Schlumberger's Software Improvement Program', IEEE Transactions on Software Engineering, Vol. 20, No. 11, November 1994, 833-839.

[YOUR95] Yourdon, Ed. '"Good Enough" Software', Software Quality Management, Issue 27, Autumn 1995, 30-33.

Appendix A

FAULT REPORT FORM

Project/System Title :

Date of Fault Occurrence : **Fault Id. :**

Fault Description :

Fault Uncovered By :

Date of Fault Repair :

Description of Fault Repair :

Modules Amended : **Screens Amended :**

Signed By :

Appendix B

CHANGE REQUEST FORM

Project/System Title :

User Name :

Date of Change Request : **Change Request Id. :**

Description of Change :

Reason For Change :

System Error/Enhancement: **Date of Change :**

Description of Change :

Modules Amended/Added : **Screens Amended/Added :**

Signed By :

Appendix C - Samples of the forms used in the PSP

Time Recording Log

Student ______________________ Date ______________

Instructor ______________________ Program # ______________

Date	Start	Stop	Interruption Time	Delta Time	Phase	Comments

Defect Recording Log

Student ______________________ Date ______________

Instructor ______________________ Program # ______________

Date	Number	Type	Inject	Remove	Fix Time	Fix Defect	Description

PSP0 Project Plan Summary

Student ____________________ Date ____________

Program ____________________ Program # ____________

Instructor ____________________ Language ____________

Time in Phase (min.)	**Plan**	**Actual**	**To Date**	**To Date %**
Planning				
Design				
Code				
Compile				
Test				
Postmortem				
Total				
Defects Injected				
Planning				
Design				
Code				
Compile				
Test				
Total Development				
Defects Removed				
Planning				
Design				
Code				
Compile				
Test				
Total Development				

PSP0.1 Project Plan Summary

Student ______________________ Date ______________

Program ______________________ Program # ______________

Instructor ______________________ Language ______________

Program Size (LOC)	Plan	Actual	To Date	To Date %
Base (B)				
Deleted (D)		______	______	
Modified (M)		______	______	
Added (A)		______	______	
Reused (R)		______	______	
Total New and Changed (N)	____	______	______	
Total LOC		______	______	
Total New Reused		______	______	
Time in Phase (min.)				
Planning	____	______	______	________
Design	____	______	______	________
.....				
.....				
Defects Injected				
Planning		______	______	________
Design		______	______	________
.....				
.....				
Defects Removed				
Planning		______	______	________
Design		______	______	________
.....				

APPENDIX D Software Process Questionnaire

Completing the Questionnaire

1. To the right of each question there are boxes representing the four possible responses to the question.

Answer **YES** when the practice is :

- Well established and consistently performed. (The practice should be performed on every occasion to meet the criteria).

Answer **NO** when the practice is :

- Not well established or inconsistently performed.

Answer **DON'T KNOW** when you have insufficient knowledge about certain aspects of your process or project to answer the question.

Answer **DOES NOT APPLY** when you have sufficient knowledge about the process or project but feel the question does not apply or has no relevance in this instance.

2. Comments should be used for the following :

- To record any supporting material to justify a **Yes** answer.
- Where a **No** is recorded but some elements of the practice are performed.
- For elaborating on **Does Not Apply** answers.
- For documenting practices performed by the company which may not be covered sufficiently by the other questions.

3. The **'You'** in the questions refers to the company or project not the specific individual.

PRELIMINARY QUESTIONS

	Yes	No	Don't Know	Does Not Apply
Software Initiatives				
1. Are all software initiatives generated from within the company? Comments:	□	□	□	□
2. Is a formal process used to evaluate each system proposal before commencing development? Comments:	□	□	□	□
Review and Sign-Off				
1. Does senior management have a mechanism for the regular review of the status of software development projects? Comments:	□	□	□	□
2. Do software development line managers sign-off schedules and deliverables? Comments:	□	□	□	□
Software Quality Assurance				
1. Does the company have a Software Quality Assurance Team? Comments:	□	□	□	□
(Onlv if 'Yes' to 1.)				
2. Does the Software Quality Assurance team have a management reporting channel separate from the software development project management? Comments:	□	□	□	□

	Yes	No	Don't Know	Does Not Apply
Configuration Management				
1. Does the company have a software Configuration Management function? Comments:	□	□	□	□
2. Is a mechanism used for controlling changes to the software requirements? Comments:	□	□	□	□
3. Is a mechanism used for controlling changes to the code ? (Who can make changes and under what circumstances)? Comments:	□	□	□	□
Estimating				
1. Is a formal procedure used to make estimates of software cost? Comments:	□	□	□	□
2. Is a formal procedure used to make estimates of software size? Comments:	□	□	□	□
3. Is a formal procedure used to make estimates of software development schedules? Comments:	□	□	□	□
Metrics				
1. Are statistics on software code and test errors gathered? Comments:	□	□	□	□

SOFTWARE PROCESS OVERVIEW

ORGANISATION FOCUS

	Yes	No	Don't Know	Does Not Apply
1. Has your organisation developed, and does it maintain, a standard software process? Comments:	□	□	□	□
If 'Yes' go to 2 otherwise go to Education and Training)				
2. Do software development staff refer to the organisation defined software process when undertaking software development? Comments:	□	□	□	□
3. Does the organisation have a Software Engineering team or individual responsible for improving the organisation's software process? Comments:	□	□	□	□
4. Do the individuals involved receive the required training to perform these activities? Comments:	□	□	□	□
5. Is the organisation's software process reviewed on a regular basis? Comments:	□	□	□	□
6. Are senior management involved in improving the organisation's software process? Comments:	□	□	□	□

	Yes	No	Don't Know	Does Not Apply
7. Is the organisation's software process documented? Comments:	□	□	□	□

EDUCATION AND TRAINING

	Yes	No	Don't Know	Does Not Apply
1. Are training activities planned? Comments:	□	□	□	□
2. Do software managerial and development staff receive the training necessary to perform their roles? Comments:	□	□	□	□
3. Are adequate resources provided to implement the organisation's training programme? Comments:	□	□	□	□
4. Are training programme activities regularly reviewed? Comments:	□	□	□	□

PROJECT MANAGEMENT

Project Planning

	Yes	No	Don't Know	Does Not Apply
1. Does a written procedure exist for managing a software project? Comments:	□	□	□	□
2. Does the project follow a written organisational procedure for planning a software project? Comments:	□	□	□	□
3. Is agreement reached in advance of development between all groups in response to their roles and commitment in relation to the project? Comments:	□	□	□	□
4. Do the software plans document the activities to be performed during the project and the deliverables for each project stage? Comments:	□	□	□	□
5. Are estimates (e.g. schedule, size and cost) made in advance of the project? Comments:	□	□	□	□

Project Monitoring

	Yes	No	Don't Know	Does Not Apply
1. Does a written procedure exist for monitoring a software project? Comments:	□	□	□	□
2. Are the actual results of the project compared with the estimates? Comments:	□	□	□	□

	Yes	**No**	**Don't Know**	**Does Not Apply**
3. Is action taken when actual results deviate from the project's scheduled results?	□	□	□	□

Comments:

CONFIGURATION MANAGEMENT

	Yes	No	Don't Know	Does Not Apply
1. Does the organisation have a documented procedure for Configuration Management? Comments:	□	□	□	□
2. Does the organisation have a documented procedure for ensuring that any software changes are reflected in the appropriate documentation at every life-cycle phase? Comments:	□	□	□	□
3. Does the organisation have a team or individual responsible for controlling changes to Software and issuing software releases? Comments:	□	□	□	□
4. Are project personnel trained to perform the software configuration management activities for which they are responsible? Comments:	□	□	□	□
5. Are change requests and error reports filed with copies distributed to affected individuals? Comments:	□	□	□	□
6. Are software changes agreed to by all affected groups and individuals? Comments:	□	□	□	□
7. Are all new software releases fully regression tested before release? Comments:	□	□	□	□

QUALITY ASSURANCE

	Yes	No	Don't Know	Does Not Apply
1. Does the organisation have a documented procedure for implementing Software Quality Assurance? Comments:	□	□	□	□
2. Does the organisation have a team or individual responsible for controlling changes to Software and issuing software releases? Comments:	□	□	□	□
3. Does the organisation specify measurable goals (e.g. reliability, portability, functionality) for managing the quality of its software products? Comments:	□	□	□	□
If 'Yes' to 3.				
4. Does the organisation compare the software output against the objective software measures? Comments:	□	□	□	□
5. Does the SQA function provide objective verification that software products and activities conform to company standards and procedures? Comments:	□	□	□	□
6. Are metrics collected during the software development phase? Comments:	□	□	□	□

	Yes	No	Don't Know	Does Not Apply
REQUIREMENTS SPECIFICATION				
1. As Requirements change are the necessary amendments made to plans, estimates and documentation? Comments:	□	□	□	□
2. Does a written procedure exist for documenting requirements? Comments:	□	□	□	□
3. Is the requirements specification/definition activity subject to SQA review? Comments:	□	□	□	□

SYSTEM DESIGN

	Yes	No	Don't Know	Does Not Apply
1. As design changes are made are the necessary amendments made to plans, estimates and documentation? Comments:	□	□	□	□
2. Does a written procedure exist for documenting design? Comments:	□	□	□	□
3. Is the design activity subject to SQA review? Comments:	□	□	□	□

IMPLEMENTATION

	Yes	No	Don't Know	Does Not Apply
1. As design defects are discovered are the necessary amendments made to plans, estimates and documentation? Comments:	□	□	□	□
2. Does a written procedure exist for documenting programs? Comments:	□	□	□	□
3. Is the programming activity subject to SQA review? Comments:	□	□	□	□

TESTING

	Yes	No	Don't Know	Does Not Apply
1. As design defects are discovered are the necessary amendments made to plans, estimates and documentation? Comments:	□	□	□	□
2. Does a written procedure exist for performing software testing? Comments:	□	□	□	□
3. Is the testing activity subject to SQA review? Comments:	□	□	□	□

OPERATIONS/MAINTENANCE

	Yes	No	Don't Know	Does Not Apply
1. As system defects are discovered are the necessary amendments made to documentation? Comments:	□	□	□	□
2. Does a written procedure exist for documenting maintenance activity? Comments:	□	□	□	□
3. Is the maintenance activity subject to SQA review? Comments:	□	□	□	□