A quality software process for rapid application development

GERRY COLEMAN\(1\) and RENAAT VERBRUGGEN\(2\)

\(1\)Centre for Software Engineering and \(2\)Dublin City University

Received March 1998 accepted April 1998

Software organizations can significantly improve the quality of their output if they have a defined and documented software process, together with the appropriate techniques and tools to measure its effectiveness. Without a defined process it is impossible to measure success or focus on how development capability can be enhanced. To date, a number of software process improvement frameworks have been developed and implemented. However, most of these models have been targeted at large-scale producers. Furthermore, they have applied to companies who use traditional development techniques. Smaller companies and those operating in development areas where speed of delivery is paramount have not, as yet, had process improvement paradigms available for adoption.

This study examined the software process in a small company and emerged with the recommendation of the use of the Dynamic Systems Development Method (DSDM) and the Personal Software Process (PSP) for achieving software process improvement.


Keywords: Software, Process Improvement, Personal, Rapid Application Development, Defect, Metrics, Reviews

1. Introduction

In order to improve its software capability each organization must have a defined and documented software process. Defining and documenting the software process allows companies to measure current performance, identify areas of weakness and initiate improvement actions. In 1988 the Software Engineering Institute (SEI) developed a Maturity Framework to appraise the maturity of software processes within companies [1]. This model indicates that the more mature the development process of the organization, the more capable it is of developing high-quality software. The Maturity Framework included process assessment methods, software capability evaluation, and a maturity questionnaire. After extensive work in this area the SEI evolved the Maturity Framework into the Capability Maturity Model (CMM) [2]. The CMM is based on the knowledge acquired from studies using the Maturity Framework and specifies recommended practices in the particular areas that have been shown to enhance software development and maintenance capability.

The CMM, however, is primarily targeted at large organizations. Small companies also face difficulties in software development and defining appropriate process models. This
study examined the problems facing small software companies and proposes using both the Dynamic Systems Development Method (DSDM) [3] and the Personal Software Process (PSP) [4] as a means to process improvement within small companies.

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

2.1 Rapid Application Development (RAD)

The term Rapid Application Development or RAD is taken to relate to projects based around tight timescales, which use prototyping and combine high-level development tools and techniques [5].

Proponents of RAD claim that it increases productivity, reduces delivery time and gains high usage because of the extent of user involvement in the development [6–9]. However RAD projects can fail because the following are not properly addressed:

- picking the right team;
- management and customer support of the project;
- the methodologies used [10].

Analysis of the material published on RAD reveals the advantages and disadvantages listed in Table 2.1.

<table>
<thead>
<tr>
<th>Advantages</th>
<th>Disadvantages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ease of implementation</td>
<td>Speed of development may result in a poorly designed product</td>
</tr>
<tr>
<td>Improved user satisfaction</td>
<td>Need more experienced development staff</td>
</tr>
<tr>
<td>Shorter time-to market</td>
<td>Strong project management and control required</td>
</tr>
</tbody>
</table>

To capitalize on the benefits listed in Table 2.1 and to address the potential problems in using RAD, the Dynamic Systems Development Method (DSDM) Consortium was established.

2.2 Dynamic Systems Development Method – a RAD standard

The Dynamic Systems Development Method (DSDM) was created in February 1995. The objective was to create a quality-centred method within which RAD techniques could be used. DSDM uses prototyping techniques to ensure the frequent delivery of software products during development. These products are delivered within fixed timescales known as ‘timeboxes’. The principles on which the method is based are listed in Table 2.2.
Table 2.2. Principles of DSDM

<table>
<thead>
<tr>
<th>DSDM principles</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Active user involvement, throughout system development, is imperative</td>
</tr>
<tr>
<td>2. DSDM teams must have the power to make decisions regarding the system</td>
</tr>
<tr>
<td>3. DSDM is focused on the frequent delivery of products</td>
</tr>
<tr>
<td>4. The primary system acceptance criterion is ‘fitness for purpose’</td>
</tr>
<tr>
<td>5. Testing is integrated throughout the life-cycle</td>
</tr>
</tbody>
</table>

2.3 The DSDM life-cycle

The development life-cycle is divided into five phases:

- Feasibility Study
- Business Study
- Functional Model Iteration
- Design and Build Iteration
- Implementation.

An overview of the life-cycle appears in Fig. 2.1.

The first phase, the Feasibility Study, determines the feasibility of the project and its suitability for development using DSDM. The Business Study defines the high-level functionality and the affected business areas.

These are then baselined as the high-level requirements together with the primary non-functional requirements. The main part of the development is contained within the two iterative prototyping cycles.

The RAD process in DSDM

![Fig. 2.1. DSDM life-cycle](image-url)
The objective of the Functional Model Iteration is to elicit requirements while the emphasis in the Design and Build Iteration is on ensuring that the prototypes meet pre-defined quality criteria. The final phase, the Implementation phase, is the handover to users, which will normally be accompanied by a project review.

2.4 DSDM – A way forward for RAD

Given that there are still fears that RAD produces poor quality, unmaintainable code, there is a definite need for a methodology which will allow quality to be incorporated into RAD projects. DSDM suggests ensuring quality products through:

- inspections, reviews and walkthroughs,
- demonstrations to user groups, and
- testing.

While there is a comprehensive account, in the DSDM documentation, of how testing should be conducted at every stage of the development process very little space is devoted to the use of alternative measures of assuring the quality of the developed product. Though the consortium admit that, ‘testing can never prove that a system works’, limited detail is provided in the documentation as to how alternative quality mechanisms, such as, inspections and reviews, can be used.

A large body of work exists [11–15] which illustrates how inspections and reviews are superior to testing at discovering errors.

The benefits inspections have had in one large company are shown in Table 2.3.

<table>
<thead>
<tr>
<th>Data category</th>
<th>1990</th>
<th>1991</th>
<th>1992</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code-inspection meetings</td>
<td>1500</td>
<td>2431</td>
<td>2823</td>
</tr>
<tr>
<td>Document-inspection meetings</td>
<td>54</td>
<td>257</td>
<td>348</td>
</tr>
<tr>
<td>Design-document pages inspected</td>
<td>1194</td>
<td>5419</td>
<td>6870</td>
</tr>
<tr>
<td>Defects removed</td>
<td>2205</td>
<td>3703</td>
<td>5649</td>
</tr>
</tbody>
</table>

Backed up by this evidence, it is essential that inspections and reviews form an integral part of DSDM.

Although the consortium state that a formal quality inspection is necessary on the Business Area Definition and the Prioritized Functions, if inspections are carried out only on these areas then testing is being used as the sole mechanism to find defects in the subsequent coding phases. The Functional Prototypes and the Design Prototypes should also be subject to code inspections to ensure maximum defect detection and conformance to requirements.

While the quality approaches in DSDM mirror the best practices in traditional software development there is a greater emphasis on the use of software tools to support the quality process and ensure development momentum. Tools such as those that provide ‘capture/playback’ testing facilities are imperative as regression testing will occur as new system increments are released.

Finally, the DSDM authors state that every DSDM project should have an accompanying Quality Plan that states how quality and standards are to be enforced.
3. Process improvement using the Personal Software Process (PSP)

3.1 Personal Software Process (PSP)

The Personal Software Process (PSP) is an attempt to scale down current software quality and assessment practices, for the improvement of individual software developers [4]. It is essentially a bottom-up approach where individuals manage and assess their own work and, as such, is of particular interest to small software houses where tailoring large-scale practices can cause difficulties.

The PSP is essentially a framework of forms, guidelines and procedures, which in effect produce a defined process. Using the PSP data gathering mechanisms provides historical data which helps you measure your performance, your work patterns and practices. By examining these factors Humphrey [4] believes the developer can:

- make better plans and more accurate estimates;
- improve productivity;
- improve product quality.

3.2 PSP improvement phases

The evolution of the various PSP phases is illustrated in Fig. 3.1.

![Fig. 3.1. The PSP evolution](image)
3.3 **PSP0 (The Baseline Personal Process)**

The principal objective of PSP0 is to provide a framework for gathering your own initial process data.

PSP0.1 extends PSP0 by the inclusion of additional planning and size measurement details. Size can be measured by counting Lines of Code (LOC), Function Points (FPs), Objects or some other suitable unit.

3.4 **PSP1 (Planning, Scheduling and Estimation)**

In order to assist with size estimation, Humphrey [4] proposes the use of the PROBE (PROxy-Based Estimating) method. A proxy is a substitute or stand-in and in this instance the proxy is used to relate product size to the functions the estimator can visualize and describe. Examples of proxies include objects, screens, files, scripts or function points. PSP1.1 also assists with task and schedule planning.

3.5 **PSP2 (Software Quality Management)**

PSP2 and 2.1 use reviews and verification methods as techniques to ensure process and product quality.

3.6 **PSP3 (Cyclic Development)**

PSP3 proposes decomposing large systems to allow PSP techniques to be used effectively.

3.7 **Personal Software Process – an analysis**

Some of the disciplines advocated in the PSP have already been implemented within many companies. Inspections, code coverage, and testing have been used to uncover errors. The cost of fixing a defect when software is in use compared to fixing it if found during coding ($1000 vs $1) can be exorbitant [16]. Therefore, using code reviews, both for original code and, importantly, for subsequently amended code can result in large cost savings.

The quality of the software product is an issue facing the majority of companies. Most have tackled it by concentrating on the quality of the software process believing that a good product consistently emerging from an ad hoc process is a remote possibility. This study has already shown how software inspections and reviews have helped a number of companies to find and fix errors earlier in the development life-cycle, when it is cheaper to do so. It has also been shown how, through using inspections, some companies have witnessed productivity increases with software being released earlier and significantly less rework.

Also software metrics are being increasingly used as a way of measuring and controlling the software process. Organizations have found that in the absence of metrics they are unaware of the state of their own process. Collecting even basic defect metrics has, at least, allowed them a preliminary evaluation of their own software capability.

From the point of view of a small software development company the PSP is particularly attractive. As with DSDM it provides a defined and documented software process. Also,
small companies will inevitably have small development teams and this will assist greatly with the implementation of PSP. Even one person in a small team using PSP can have a noticeably greater impact on the resultant product quality than one person in a large team. There is also the potential for such a person to act as a ‘process champion’ with other team members adopting the approaches.

4. Assessment of the software process within a small company

Without real data on how small companies operate, it is difficult to determine which process models are appropriate and how they should be implemented. The purpose of this section is to examine how software is developed in a small software company.

This was not intended as a software capability assessment but as a first step to gather information about how the company’s software process functioned. The company examination is divided into two sections. Firstly, a questionnaire was used to obtain general information on the company’s processes. Secondly, having processed the questionnaire responses, a detailed evaluation of the company was then undertaken. This is covered in Section 5.

4.1 Questionnaire findings

The company was assessed under twelve headings.

The figures involved represent the percentage achievement of target. Target is the carrying out of all the activities, represented under each heading, at all times. Target is 100%.

The questionnaire results are represented in Figure 4.1.

The results show some fundamental process weaknesses, so to get a more detailed picture of how the company develops software a detailed evaluation was then conducted.

5. Evaluation of the software process within a small software company

5.1 Evaluation findings

The evaluation of the organization’s development process highlighted some evident strengths. These included the fact that users were heavily involved in systems development, up-to-date tools were being used very effectively and developers were highly skilled.

However, more fundamental weaknesses were also apparent. Apart from the absence of a defined software development process the company lacks formality and standards in the areas outlined in Table 5.1.
Fig. 4.1. Company process maturity chart

Table 5.1. Company process weaknesses in standardization and procedures

*Fundamental development process weaknesses*

1. No standard user requirements document
2. No standard design document
3. No programming standards exist
4. Programmers are not required to produce unit test plans
5. No formal independent testing of modules
6. No formal documentation of errors found during acceptance testing
7. No recording of live fault reports and change requests from users

6. A software process for RAD

The purpose of this section is to propose how a combination of DSDM and PSP disciplines can be used as a solution to the software process problems highlighted in Sections 4 and 5. Throughout this section, the phrase ‘the company’ refers to the company assessed and evaluated in Sections 4 and 5.
6.1 Using DSDM within the company

There are two questions which need to be answered:

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

The points which favour the use of DSDM within the company are documented in Table 6.1.

Table 6.1. Factors which favour DSDM usage within company

<table>
<thead>
<tr>
<th>Factors favourable towards DSDM usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Highly-skilled development staff</td>
</tr>
<tr>
<td>2. Company application area (multimedia) already includes major user involvement in development</td>
</tr>
<tr>
<td>3. Prototyping techniques are currently employed</td>
</tr>
<tr>
<td>4. Expertise in latest RAD and software tools</td>
</tr>
<tr>
<td>5. Senior management are committed to improving quality</td>
</tr>
<tr>
<td>6. Desire to get products onto the market more quickly</td>
</tr>
</tbody>
</table>

6.2 Using PSP within the company

The factors which favour the introduction of PSP are outlined in Table 6.2.

Table 6.2. Factors which favour PSP usage within company

<table>
<thead>
<tr>
<th>Factors favourable towards PSP usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Commitment to quality</td>
</tr>
<tr>
<td>2. Desire for documented process and standardization</td>
</tr>
<tr>
<td>3. Desire to commence metrics gathering</td>
</tr>
<tr>
<td>4. Skilled, enthusiastic developers</td>
</tr>
<tr>
<td>5. Small development staff complement</td>
</tr>
</tbody>
</table>

6.3 A quality software process for RAD

While DSDM offers a ready-made life-cycle for RAD it is weak on the application of quality measurement techniques and metrics collection. PSP, conversely, offers process improvement at an individual level but is not RAD oriented. Taken together DSDM provides the life-cycle and framework which can be used by RAD teams and PSP offers the necessary quality control mechanisms.

6.3.1 Combining DSDM and PSP3

PSP3 offers the closest match with the objectives and framework of DSDM.

The Requirements and Planning stage, of PSP3, carries out similar functions to DSDM's Business Study and the Cyclic Development section closely matches the Design
and Build Iteration in DSDM. Where DSDM scores over the PSP, for the type of applications used by the company in this study, is the increased emphasis on iteration particularly in the Functional Model stage.

However, PSP3 includes unit and integration test within each of its development cycles. This is particularly important as each increment is adding extra functionality to the previous increment. If there are residual defects in the initial increment then these may cause a ‘ripple’ effect in subsequent increments. Furthermore, subsequent increments must also find these defects. PSP3 insists on the use of reviews to prevent this. Using reviews at each of the DSDM prototyping phases will ensure that ‘clean’ versions of software are used as input to subsequent system increments.

6.3.2 Using proxies

The DSDM consortium contend it is easier to calculate how much can be done by a certain time than to calculate how long it takes to do something; thereby promoting the use of timeboxes within which given portions of functionality can be delivered. The PSP suggests using proxies as a size estimating technique. If the proposed data collection and recording mechanisms are used, you will soon have historical data from which you can determine how many LOC or Function Points (FPs) can be developed within the timebox. In the absence of historical data and size measurements, the company is at risk of overestimating development time and therefore losing potential customers or substantially underestimating development time with the consequent failure to meet deadlines, reduced quality end products and/or substantial overtime requirements or extra staff complement. The use of proxies will help this company refine its estimates.

6.3.3 Testing

In the section of the DSDM documentation devoted to testing, the consortium state that, though testing is burdened by time and resource constraints, no amount of testing would locate all errors.

The consortium also state that during testing ‘confidence is derived from finding errors which are then fixed’. Users who have spent many excessive hours in acceptance testing would balk at this statement. Continually finding errors in products at this stage substantially reduces their confidence in the system, as they have no way of knowing how many errors they are not finding!

Without proper reviews and inspections, prior to product handover to the user, there is potential for the user to receive error-laden code. This could certainly reduce the user’s confidence in the system and generate substantial rework. In such an environment, development schedules would not be met and development time would be lengthened thus invalidating the use of RAD.

An improved procedure would be for each individual developer to conduct a design and code review with testing then executed by a technical peer. This will ensure that the product the user receives will be more error-free. This would increase user confidence in the product and allow them to concentrate their testing efforts on the major areas of system functionality.

Any moves towards improved testing procedures within the company should be accompanied by a commensurate effort to improve inspection and review techniques. The
DSDM Consortium recommend the use of static code analysers since they do ‘a degree of code inspection almost for free’. However, until the company is wholly satisfied with the specific code analysis tools, available for use with its development environment, it should concentrate on a manual inspection/review approach for code.

6.3.4 A quality plan

It is stated that every DSDM project should have a quality plan that outlines how quality will be controlled and standards applied. The proposed quality software process for RAD will have a quality plan and could include much of the documentation provided by the PSP.

For example, the PSP process scripts for each level of the PSP should be included in a quality plan.

Having a process script which outlines the Entry and Exit criteria for a phase with a detailed outline of the steps to be undertaken during that phase is a very useful quality control technique. The process scripts are accompanied by planning, development and postmortem scripts for each phase and these again ensure quality coverage of the process. Including these documents in a quality plan introduces the standards and control elements demanded by DSDM.

Another quality control mechanism which could be adapted from the PSP for use in the new environment is the checklist. Design and code review checklists, which could be established for each language used by the organization, and any documentation used with software inspections should form part of the plan. The PROBE estimating script, introduced in PSP1, could also be included in the plan.

Another PSP document which would be part of the quality plan is the Process Improvement Proposal (PIP), which is part of PSP0.1. The PIP provides a means of documenting process shortcomings and suggested solutions. PIP copies should be retained in the quality plan.

One way of assessing the effectiveness of the quality plan is to measure the process itself. The PSP provides quality measures for use in PSP projects. However, some new measures are appropriate for use in the quality software process for RAD.

6.3.5 Metrics

At present, the DSDM consortium is not recommending any specific approach towards collecting metrics. Also, what is stated, in the DSDM Manual, is that ‘during system testing, a count of detected errors should be kept with the aim of improving this as time goes on’. If a process is to be successful, and continually improve, it is too late to start collecting error metrics at the system testing stage.

There are several reasons for this. Firstly, it may be very difficult to determine the exact cause of the error as the defect which gives rise to the error may have been introduced during any previous timebox or prototyping phase. Secondly, no record will be available of the time it has taken to fix errors (which have been removed prior to system testing) and during which activity they were removed. Thirdly, there will be no picture available of how effective the interim testing is, i.e. testing of timebox elements, prototypes, etc. Also, DSDM proposes the analysis of errors and their causes. Again, without collecting error data as the project progresses, identifying error causes will be a complex and imprecise activity.
A list of the metrics proposed for use within the company appears in Table 6.3.
It is not recommended that all of these metrics should be introduced immediately into a project. It is only through using and collecting metrics that companies or individuals can decide which are the most appropriate and cost-effective to collect.

Table 6.3. Metrics which could be introduced effectively into the company

<table>
<thead>
<tr>
<th>Metric type (example)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Defect metrics (total defects per KLOC, test defects per prototype etc.)</td>
<td></td>
</tr>
<tr>
<td>2. Productivity metrics (LOC/hour, function point/day, etc.)</td>
<td></td>
</tr>
<tr>
<td>3. Size metrics (total new &amp; changed LOC, size estimation error etc.)</td>
<td></td>
</tr>
<tr>
<td>4. Time/schedule metrics (time estimation error, delivery ratio etc.)</td>
<td></td>
</tr>
<tr>
<td>5. Extended metrics (phase yields, defect removal leverage etc.)</td>
<td></td>
</tr>
<tr>
<td>6. Maintenance metrics (live/development defect ratio, defect fix time ratio etc.),</td>
<td></td>
</tr>
</tbody>
</table>

**Size and defect metrics**

Size can be counted at all stages of development and can be based on lines of code (LOC) or function points (FP) delivered. These figures can be used to determine the estimation errors in code size, i.e. percentage difference between estimated size and actual size, and can be calculated as follows:

\[
\text{error} \% = 100 \times \frac{(\text{actual size} - \text{estimated size})}{\text{estimated size}}
\]

Size estimation errors, by prototype and by timebox, should also be measured. Defects will be measured relative to code size. Measuring defects by timebox and by prototype produces the following examples:

\[
\text{total defects/size (timebox)} = \frac{\text{total defects per timebox}}{\text{size per timebox}}
\]

for assessing the effectiveness of the testing process:

\[
\text{test defects/size (prototype)} = \frac{\text{test defects per prototype}}{\text{size per prototype}}
\]

A defect database should be maintained containing information about the defects injected and removed during development phases. The database should contain fields for Defect Number, Defect Type, Inject Phase, Remove Phase and Fix Time. A Prototype Identifier could also be recorded. Additionally, new fields, such as Timebox Number and Object/Function/Method Number and Type (e.g. I/O, Interface, etc.) could be included. This extra data will assist in future defect analysis.
**Productivity metrics**

The standard productivity measures relate to code amendments and additions.

If we treat **Additions** as either new and changed lines of code (NLOC) or new and changed function points (NFP) and **Time Units** of hours in relation to NLOC and days in relation to NFP then **Productivity** can be measured as

\[
\text{Productivity} = \frac{\text{Total Additions}}{\text{Development Time Unit}}
\]

It is also recommended to measure productivity by timebox and by prototype. However, it should be noted that using RAD tools where a lot of screen design is completed without the requirement to write code, productivity figures, if based on lines of code, can look low.

Similarly, if there is significant code reuse then again productivity figures may be underestimated. The above two elements should be factored into any productivity estimates and calculations.

**Time/schedule metrics**

For future estimates of development time or deliverable capability per timebox it is necessary to collect time measures. These time measures can be used to determine how much time was spent in each of the particular development phases and should be counted in the most appropriate units, e.g. hours, days, weeks, etc. If the productivity measures are to be accurate, then time must also be recorded per prototype and per timebox. Time estimation error, i.e. percentage difference between estimated time and actual time, can be calculated as follows:

\[
\text{error} \% = 100 \times \frac{\text{actual time} - \text{estimated time}}{\text{estimated time}}
\]

However, using a formula similar to the one for error estimation can assist in measuring the closeness between predicted timebox deliverables and actual timebox deliverables. As the prime objective in DSDM is ‘building the right system’ this metric will serve as both a quality and productivity measure for the development. Assuming this measure is termed the delivery ratio (DR) it can be calculated thus:

\[
\text{DR (total development)} = 100 \times \frac{\text{delivered functionality}}{\text{planned functionality}}
\]

This should also be calculated by prototype and timebox. Also, the measure could be used at the end of any iterative phase or timebox to determine the ratio of delivered functionality to date:

\[
\text{DR (milestone A)} = 100 \times \frac{\text{delivered functionality to milestone A}}{\text{planned functionality to milestone A}}
\]

**Extended measures**

With the basic metrics in place, some derived measures can also be generated.
The ‘yield’ of a phase is the percentage of defects removed in a phase over the total number of defects removed and remaining in the product.

For example, a prototype yield could be:

\[
\text{yield (prototype)} = 100 \times \frac{\text{defects found in prototype}}{\text{defects found in prototype + escaping from the prototype}}
\]

This measure will be useful in determining the quality of the defect detection mechanisms used.

The defect removal leverage (DRL) provides a measure of the effectiveness of different defect removal methods. The DRL is the ratio of the defects removed per hour in any two phases and is particularly useful in comparing say a review phase with a test phase. These approaches could again be profitably employed in DSDM by measuring defect removal rates in prototypes and timeboxes.

For example, the measures could be calculated for a prototype as:

\[
\text{DRL (code review = defects/hour (code review for prototype A)} \quad \text{for prototype A)} \quad \text{defects/hour (unit test for prototype A)}
\]

High review yields are preferable to high test yields as they show that defects are being found earlier in the process and are thus cheaper to correct.

6.3.6 Maintenance

Projects developed in RAD environments have, in the past, been criticized for being unmaintainable. The quality software process for RAD proposed in this study will improve the maintainability of software through improving quality. To measure maintenance effort post-delivery defects should be collected. This measure, the live to development defect ratio (LDDR), could be used to determine the ratio of defects found in the live environment to those found in development:

\[
\text{LDDR} = 100 \times \frac{\text{defects found in live environment}}{\text{defects found during development}}
\]

Another complementary maintainability measure would be the time spent fixing defects after development, calculated as the average defect fix time post-delivery versus the average fix time prior to delivery. The average defect fix time during development could be calculated thus:

\[
\text{avg. defect fix time (during development)} = \frac{\text{total defect fix time during development}}{\text{total number of defects during development}}
\]

The average defect fix time post development would be:

\[
\text{avg. defect fix time (post development)} = \frac{\text{total defect fix time post-development}}{\text{total number of defects post-development}}
\]
Therefore, the defect fix time ratio (DFTR) for two phases would be:

\[
\text{DFTR (phase A /phase B)} = \frac{\text{average defect fix time (phase A)}}{\text{average defect fix time (phase B)}}
\]

For post-delivery versus pre-delivery this would be:

\[
\text{DFTR (post-delivery / pre-delivery)} = \frac{\text{average defect fix time (post-delivery)}}{\text{average defect fix time (pre-delivery)}}
\]

These metrics are very significant in that they show clearly the time and cost implications of not catching defects early when they are cheaper to fix.

6.3.7 Software reuse

Both DSDM and PSP state that reusing existing software components will result in productivity and quality improvements. There is no doubt that there are significant productivity benefits to reusing software. There are, however, significant costs associated with reuse. Apart from the extra time it takes to design and develop reusable parts, there is the question of managing and maintaining the reuse library. A DSDM project which possesses tight delivery deadlines may not have the time to develop reusable parts. They may have to be re-engineered later. There is always the possibility though that this re-engineering may then necessitate and trigger other re-engineering requirements within the same project. It is up to the company, implementing the PSP and DSDM to decide what its own policy is on reuse and where the costs are to be borne. Ultimately, from a corporate perspective, there is no benefit to developing reusable components if the procedures and personnel are not in place to manage and control software reuse component libraries.

7. Conclusions

7.1 Detailed analysis

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

DSDM offers the potential for faster delivery of software. To ensure that the software is of sufficient quality requires quality control and assurance mechanisms, such as checklists, reviews and inspections, associated with PSP to be used. Further benefits of using the PSP with DSDM include:

- PSP3 supports unit and integration testing during each iterative cycle.
- Proxy-based estimating will help with defining timebox elements in DSDM.
- PSP supplies appropriate metric-collection techniques that can be used in DSDM projects.
- The documents and scripts associated with PSP can be used in a quality plan.
7.2 Recommendations

7.2.1 Data recording assistance

There is no doubt that the detailed form filling required to gather all of the necessary data for process measurement within the PSP will hinder the speed of development in the initial phases. In order to prevent this, organizations using PSP measures within a RAD framework could either simplify the recording approaches or investigate the use of software tools to collect some of the basic data. One approach would be for developers to maintain detailed logs of their time on randomly selected days only [17]. With respect to using software tools to reduce the data collection overhead, there are a number of tools available to count lines of code (LOC). In areas where the user interface is crucial, LOC counting may not be as effective as function point counting. Tools which can count function points after development would be of tremendous benefit in such an environment.

Similarly, tools which assist in defect and time recording would be advantageous.

7.2.2 Inspection/review assistance

DSDM recommends the use of automated support for code reviewing and inspection. Particular benefit will be gained if these tools can be adapted for use with the PSP. Ideally then, they could not only highlight code errors but by reference to, perhaps, the PSP defect standard, they could record the category of defect. These tools, however, should only be considered when the manual review and inspection process is fully understood.

References