"SLCM: A Software Lifecycle Model to support SPI and Software Metrics"

NOTE: this paper has been submitted for publication and is under review. As such, the concept status applies. When making references to this article, please use the current title, name of author and date. Organisations willing to use this model are kindly requested to contact the author.

Hans Sassenburg [1]

Abstract
In the software industry, software process improvement (SPI) and software metrics have received considerable attention over the last decade. Several process models and standards have become available to support process thinking, process improvements and quantitative decision-making. Some parts of the software industry have been successful in adopting these models and standards to improve their software capabilities. Processes have been defined and institutionalised; process and product measurements are used to support proactive decision-making when controlling software projects. However, most software organisations are still relatively immature and struggle to successfully implement sustainable improvements. SPI initiatives are started, halted, restarted and eventually stopped. Metrics programmes are often started in isolation from SPI initiatives and are tolerated as long as they do not cost too much and do not hamper the normal operation. Failing to mature the software industry is becoming increasingly dangerous as software is playing a growingly important role in today's society. Failing to make sustainable improvements is often considered to be the result of insufficient management awareness and commitment. This might be true, as managers in general tend to ask questions like: what is the return on investment of this improvement or does it make my current projects better and faster? It is very difficult or even impossible to answer this question when there is no solid process foundation. It simply means that measuring any improvements is impossible, because it is totally unclear what to measure, how to measure and when to measure. In this article a software lifecycle model is proposed, laying the foundation for future SPI and metrics initiatives. It is argued that each professional software organisation must have such a foundation a place before questioning the return on investment of improvement initiatives. As such, it helps relatively immature organisations to focus on the implementation of necessary measurements during the entire product lifecycle. By doing so, it is inevitable that certain processes must be defined and implemented. This is a non-process and non-standard driven approach that might help organisations to implement SPI and metrics initiatives in an indirect way from a different perspective. Further, the focus is not restricted to product development only but includes product maintenance as well. This might help those organisations, that are looking for ways to bridge the gap between development and maintenance activities. The ultimate goal is not to reach a maturity level of a process model or meeting the requirements of a process standard, but to lay the necessary foundation for controlling product development and maintenance by using quantitative information. Organisations having this foundation in place will have the ability to discuss possible improvements based on quantitative information. The proposed software lifecycle model ***SLCM*** will offer the possibilities to derive those improvement initiatives that have potentially the highest return on investment.

Keywords: Life-cycle management, software process improvement, software metrics, Personal Software Process, size estimation, time estimation, effort estimation, defect detection, defect removal, yield, appraisal/failure ratio, productivity, reliability.

1. Introduction

The large application of information technology has an enormous impact on society and as a result the software industry has become critical. The fact is that the development of software products is often performed in an 'ad hoc' way. The Capability Maturity Model® or CMM® as developed by the Software Engineering Institute (SEI) defines five different maturity levels for the development process of a software supplier [SEI 1995]. The SEI publishes twice a year a *Maturity Profile Update*. These

[1] *Hans Sassenburg is Managing Director of SE-CURE AG, P.O.Box 340, 3775 Lenk, Switzerland. He is also conducting a PhD research at the University of Groningen (The Netherlands), investigating how the release decision-making process of software products can be improved. E-mail: hsassenburg@se-cure.ch.*
® *Capability Maturity Model and CMM are registered in the U.S. Patent and Trademark Office.*

profiles list the percentage of officially assessed software suppliers performing at each level. [2] At the end of 2002 the percentage of assessed suppliers still performing at the lowest maturity level equalled 24.8% [SEI 2002].[3] This simply means that these software suppliers have the most important processes with regard to project management not in place. It is commonly assumed, that the assessment of all software suppliers worldwide would show a far more dramatic picture. Estimations are that 85% to 95% of all software suppliers do not meet the criteria of CMM level 2.

The CMM and its successor CMMI[sm] have been widely accepted as a process model to steer software process improvement (SPI) activities. In some parts of software industry, like the defense industry, software suppliers are obliged to have their software processes at least at CMM level 2 or 3 in order to get any contracts. In many organisations, software has become such a critical part in the product-portfolio, that senior management at the strategic level has defined objectives and allocated budgets for improvements. Despite these successes to realise sustainable improvements, the percentage of failures is however probably some magnitudes higher. Despite large investments in process improvements, many software organisations still reside firmly at CMM level 1. As a consequence, any initiative is likely to fail as the attitude of people involved may best be described with the acronym BOHICA. [4] Many studies have revealed critical success factors to successfully implement these SPI initiatives, where management awareness and commitment are considered to be crucial ones [ZAH 1998]. However, two important factors might be missing in these studies. In the first place, it is often not the lack of management awareness and commitment but the lack of knowing how to select reasonable improvements with confidence that the money invested will be paid in a given timeframe. Managers are faced with a difficult job here. There is still little evidence that conformance to process models and standards guarantees good products, which meet customers' demands [KIT 1996]. Secondly, projects in CMM level 1 organisations are often characterised by budget and time pressure, especially when they are progressing to the set release date. This hampers improvement initiatives, as the focus is inevitably on getting the work done instead of having philosophical discussions about management policies and new working methods. This might even happen again in CMM level 2 or higher organisations, when the focus is on cutting down budgets and meeting the promised release date.

Another approach to make a breakthrough in these situations might be to address SPI initiatives from a different point of view, leaving out or changing the name in order to avoid resistance. In this article such a different approach is proposed. The approach focuses on the lifecycle modelling of software products, where both the development and maintenance phase are taken into account. This lifecycle approach incorporates many metrics that are defined by Humphrey in his Personal Software Process[sm] or PSP[sm]. First, an overview of these metrics will be given, including a discussion of tradeoffs between different metrics with regard to their optimal values. This is covered in sections 2 and 3. Hereafter, a lifecycle approach to software product development and maintenance is discussed in section 4, followed by revealing the results of empirical research in seven different European organisations in section 5. The proposed lifecycle model and an explanation of the underlying metrics are presented in section 6 and 7 respectively. Some remarks about how to use the model will be discussed in section 8. Finally, in section 9 the main conclusions are drawn with a presentation of future research steps.

2. Overview of the Personal Software Process

Humprehy introduced the PSP in 1995 [HUM 1995] and published a summarised version in 1997 [HUM 1997]. The development of the PSP was a result of the success of the Capability Maturity Model or CMM [SEI 1995]. By starting the improvement of processes at the level of projects and organisations using the CMM, room was created to address the process capabilities of individual software engineers as well. The PSP is taught in an intensive course of two weeks, where software engineers are confronted with their personal way of developing software. After having baselined their current performance, they are taught in small steps how insight can be gained in their personal software process. Further, by changing their processes they are taught how to improve the predictability (better

---

[2] *"Officially" means here that the assessors have followed special courses organised by the SEI and that assessment results have been passed forward to the SEI.*

[3] *Currently, a transition takes place from the CMM to the CMMI ('I' stands for 'Integration'). The SEI has announced in early 2002 that support for the CMM will be stopped from 2004 on.*

[4] *"BOHICA" stands for "Bend Over, Here It Comes Again".*

[sm] *CMMI, CMM Integration, Personal Software Process and PSP are service marks of Carnegie Mellon University.*

estimates), the effectiveness (higher quality) and the efficiency (higher productivity) of their work. The ultimate goal is to produce high-quality software.

In Figure 1, the process flow of the PSP is shown. It is based on [HUM 1997, p. 125] with only minor changes. This process flow was developed for the Eindhoven University of Technology (The Netherlands), where PSP was successfully introduced as a separate course in the post-masters education programme OOTI (1998).

The *Process Script* defines which activities have to be performed in each development phase. In the first phase *Analysis* the *Requirements* are studied, where after estimates are made for size of the resulting program, the productivity, the effort distribution over the subsequent phases, and other product and process metrics. These estimates are recorded on the *Project Plan Summary*. During the subsequent phases, the actual *defect and time data* are logged on the *Defect Log Form* and the *Time Log Form*. The *Design Review Checklist* and the *Code Review Checklist* are used to structure the review process during the *Design Review* phase and the Code *Review phase*. When the program has been tested and is ready to be released as a *Finished Product*, all recorded data is summarised on the *Project Plan Summary*, enabling the software engineer to make an analysis in the *Post Mortem* phase prior to the program release. The purpose of this analysis is to investigate how the process used can be further improved. The result may be that the *Process Script,* the *Design Review Checklist* and/or the *Code Review Checklist* are adjusted. Further, the data from the *Project Plan Summary* can be used to calculate accumulated values for productivity, time or effort distribution, and so on. This information can be stored in a *Repository* to support the estimation process for future programming assignments.

<Figure 1 to be inserted here>

By following the PSP process, valuable product and process data are obtained. Analysis of the data enables a software engineer to calculate for instance the following metrics:

- ***Estimation Accuracy***. Estimates are collected for the size of the program to be developed and the process to be followed (time distribution, productivity, etcetera). These estimates are compared with the actual figures when the program has been implemented and tested. The result is the estimation inaccuracy, which can be calculated with the following equation:

$$\text{Estimation Inaccuracy} = \frac{\text{Actual Value - Estimated Value}}{\text{Estimated Value}} \times 100\% \qquad \textbf{(1)}$$

- ***Relating Size and Effort: Productivity***. By measuring the size and effort of programs, the productivity can be calculated in retrospect. When the productivity has been measured however, it can be used to proactively calculate the expected effort or time needed to develop a program when the size of the program has been estimated, the equation being:

$$\text{Calculated Effort} = \frac{\text{Estimated Size}}{\text{Measured Productivity}} \qquad \textbf{(2)}$$

- ***Yield***. When developing a program subsequent development phases are passed through. In each development phase defects will be injected and removed. Removed defects might have been injected in the current development phase, however they might also have been injected in previous development phases. The yield of a development phase can be defined as a number, expressing the number of defects removed in a particular phase divided by the sum of the number of defects inherited from previous phases and the defects injected in the phase itself [5]:

$$\text{Yield (X)} = \frac{\text{\# defects removed in phase X}}{\text{(\# defects inherited from previous phases) + (\# defects injected in phase X)}} \times 100\% \qquad \textbf{(3)}$$

  The yield of a phase can vary from 0% to 100% (if the denominator equals 0, the yield of the phase is equal to 100%). When the yield of a phase equals 0%, it means that in that particular

[5] *'#' stands for 'number of'.*

phase no defects have been removed. They are all carried forward to the next phase. When the yield of a phase equals 100%, it means that in that particular phase all existing defects are removed. These are both the defects inherited from previous phases and the defects injected in that particular phase itself. In the ideal case, the yield of each development phase is equal to 100%. This means that in each phase all injected defects are also removed and that the final product is free of defects. But in practice, not all defects will always be removed. Defects will be carried forward to the subsequent phases. If they are not found in, they will be shipped being part of the final product. Note that the yield of the different development phases can only be calculated when all defects have been found. Only then it is known how many defects where injected and removed in the different phases.
If an injected defect is carried forward to subsequent phases, it is very important to detect and remove the defect as early as possible. The rework effort is proportional to the difference between the injection phase and the detection and removal phase. If an early injected defect is found in one the latest phases of development, the rework effort will extend over more phases. Very expensive defects are the ones that are injected during the Analysis and Design phase and detected during the Test phase. [6] Another approach is now to calculate the Yield of the phases Analysis and Design combined:

$$Yield_{AD} = \frac{\#\ defects\ removed\ in\ phases\ Analysis\ and\ Design}{\#\ defects\ injected\ \ in\ phases\ Analysis\ and\ Design} \times 100\% \qquad (4)$$

This metric denotes the quality of the early phases before handing over the blueprint of the product for further implementation and testing.

- ***Appraisal/Failure Ratio (A/FR)***. Striving for a high combined Yield means investing effort in appraisal to detect defects as early as possible. An example of an appraisal action is for instance a review. The objective of appraisal actions is to pass forward a reliable product to the next phase. Advantages are twofold. In the first place this will probably lead to a more reliable final product. It is a utopia to think that all transported defects will be found at a later stage. In the second place the total failure cost in subsequent phases due to fixing detected defects will probably decrease as the number of expensive defects decreases. A new metric is introduced, called the Appraisal/Failure Ratio or A/FR. This is the ratio between the appraisal cost in the early development phases and the failure cost in subsequent development phases, its equation:

$$Appraisal/Failure\ Ratio = \frac{appraisal\ effort\ in\ phases\ Analysis\ and\ Design}{failure\ effort\ in\ subsequent\ phases} \qquad (5)$$

The A/FR can vary from 0 to infinite (if the denominator equals 0, the A/FR is equal to 0 if defects are found but not removed and the A/FR is undefined if no defects are found). An A/FR equal to 0 means that no appraisal effort is spent in detecting and removing defects in the early phases of development. The failure cost to be made in subsequent phases later will presumably be very high, assuming that the reliability of the final product is important.

- ***Defect Injection and Defect Removal Rate***. Collecting defect data enables one to calculate the defect injection and defect removal rate in each development phase. It can for instance be related to the effort spent in that particular phase:

$$Injection\ Rate\ (X) = \frac{defects\ injected\ in\ phase\ X}{effort\ in\ phase\ X} \qquad (6)$$

$$Removal\ Rate\ (X) = \frac{defects\ removed\ in\ phase\ X}{effort\ in\ phase\ X} \qquad (7)$$

- ***Defect Density***. The defect density can be calculated by relating the total number of defects found during development to the resulting program size:

[6] *It is assumed here, that there are no defects in the requirements.*

$$\text{Defect Density} = \frac{\text{\# defects found}}{\text{program size in KLOC}} \qquad \textbf{(8)}$$

A defect density equal to 0 means that no defects are introduced during development; in other words the Yield of each development phase equals 100%.

These metrics are powerful to analyse the software process and adjust the process in order to implement improvements. Most metrics are self-explaining with regard to their best values. The best values for Estimation Inaccuracy and Defect Density are for instance 0%, for Yield the best value would be 100%. This is however not evident for a metric like A/FR. But there is more. The best value does not necessarily mean that it is the optimal value. There might be practical limitations to obtain a Yield of 100% for instance. Human work is not perfect, causing the injection of defects, and a project budget is normally limited, meaning that time for appraisal and failure expenditures is limited. It is worth considing the question whether there are optimal combinations of a set of metrics.

3. Yield and A/FR: optimal values?

It was stated earlier that it is very important to detect and remove defects as early as possible. Late defect removal is more expensive. In Figure 2 it is depicted what the cost are to fix a requirements defect in later phases at project level [BOE 1981].

<Figure 2 to be inserted here>

How does the A/FR help to reduce the number of expensive defects? Increasing the value for the A/FR will have the following effect. By introducing appraisals such as reviews in the earlier development phases, the failure cost in subsequent phases will potentially decrease. In that case the reviews have a positive return on investment, because the decrease in failure cost will be more than the invested appraisal cost. This will be the case when there are many defects in a program. With little effort the Removal Rate can be substantially increased and the overall Productivity will be improved. There is however an optimum. Beyond this point, the law of diminishing returns applies. A further increase in appraisal cost will have less effect on the reduction of failure cost and the overall Productivity will decrease again. The Point of Optimality is the minimal value of the sum of the appraisal cost and failure cost:

$$\text{Point of Optimality} = \text{Min}\ (\ \text{Sum (appraisal cost, failure cost)}\ ) \qquad \textbf{(9)}$$

This has been depicted in Figure 3. By further improving the A/FR beyond this point, it is theoretically possible to reach a Yield of 100%.

<Figure 3 to be inserted here>

What is the optimal value for the A/FR? This is a question that needs to be addressed at project level in its specific business context. The answer is influenced by factors like the type of application and the market situation. It might be the case for instance, that an application demands an extreme high level of reliability. Such requirements can only be met by an extremely high Yield, as detecting defects in later development phases becomes increasingly difficult. An example is the aerospace industry. A wrongly launched rocket being destroyed will cost a fortune. In this case a high value for the A/FR is justified, although it will imply a decrease in productivity. Another example is the selling of high-volume products, requiring a high level of reliability. It is nearly impossible for a supplier to call back millions of televisions or a hundreds of thousands of cars (failure cost). On the other hand, there might also be situations where a software product supplier consciously takes a low A/FR for granted. A supplier wants for instance to introduce a new product and decides to prevail time to market above functionality and reliability. The initial appraisal and failure cost are brought down to strictly necessary values. After having gained the Early-Adopters advantage in a new market, the necessary failure cost is made afterwards although presumably some multiples higher.

So it can be concluded that the optimal value for the A/FR is influenced by the required reliability of the final product. If a product is delivered with a higher reliability than strictly needed by the customer(s), then either the appraisal cost has been too high (A/FR too high) or the failure cost has been too high (A/FR too low). Normally however, products in the software industry tend to have too many defects, leading to unwanted failures. In these cases, it is always worth striving for a higher Yield by increasing the A/FR because the Point of Optimality has not yet been reached. The overall reliability and productivity will increase: *better quality for less money in less time*.

4. Lifecycle Approach

The PSP metrics are owned and used by individual software engineers. They are private to them and are not meant to be used by (project) management. If software engineers were asked or even forced to hand over their data to management, they would probably change their data in such a way that their management would be pleased by it: very accurate estimations, a high productivity and excellent figures for Yield and A/FR. Management must not ignore this and be aware of the principle "you get what you ask for". Further, the accumulation of PSP data is not the data they should be interested to look into. The PSP data from engineers cover only a small part of the software lifecycle, namely the implementation phase when software modules have been defined on a higher abstraction level such as the software architecture. Managers should be interested in the total software lifecycle, expanding from the initial idea of a software product till its end-of-life. This covers both the initial development of the product and the operational phase, in which subsequent versions of the software product are developed and released. Research shows that up to 70% of the total lifecycle cost consists of maintenance cost after having released the first product version [McK 1984]. Activities in the maintenance or operational phase to be distinguished are [IEE 1998]:

- ***Corrective maintenance***: reactive modification of a software product performed after delivery to correct discovered faults (17% of total maintenance cost [NOS 1990]).
- ***Adaptive maintenance***: modification of a software product performed after delivery to keep the computer program usable in a changed or changing environment (18% of total maintenance cost [NOS 1990]).
- ***Perfective maintenance***: modification of a software product performed after delivery to improve performance or maintainability. The improvement of performance could be described as *evolutive maintenance* and the improvement of maintainability as *preventive maintenance* (65% of total maintenance cost [NOS 1990]).

Many software suppliers have a short-term horizon disregarding the total life-cycle effects [BER 2001]. In these cases, the focus is on controlling the cost and schedule of the current product release. This potentially leads to sub optimisation instead of a strategic long-term approach and as a consequence the premature releasing of software products and rapidly increasing maintenance cost. It might even lead to the situation where the software is no longer maintainable and must be replaced by a new generation.

5. Research Results

In 2003, empirical research has been conducted in seven European software organisations, working at different CMM levels (1, 2 and 3). One of the objectives of this research was to investigate to what extent these software organisations can predict the expected maintenance cost before releasing their software products. The main conclusions were [SAS 2003]:

- ***The final release decision was made without being able to accurately estimate operational cost***. In all cases, reliability and maintainability were defined as important non-functional requirements as they determine to a great extent the operational maintenance cost after product release. High reliability reduces corrective maintenance effort and high maintainability reduces both corrective maintenance effort and adaptive/perfective maintenance effort. In all cases, these non-functional requirements were not deployed to lower level components as identified in the selected product design or software architecture. It was only during testing that much effort was spent on meeting a high level of reliability. However, in all cases the organisations were unable to make solid statements about the reliability and maintainability in quantitative or even financial terms.
- ***There was no strong feedback loop in the product development cycle***. After product release, there were no specific actions undertaken to evaluate the result of the decision. In only one case there was a plan to evaluate the business case used as the rationale for the project at predefined moments

after product release by the business project leader, who was assigned the responsibility for the investments made. Further, in all cases there was no system in place to analyse the reported failures found after product release and use the results to remove process deficiencies in product development.

As a consequence, the following risks surface:

- ***Unpredictable product behaviour***. It is very difficult if not impossible to guarantee the end-user(s) what the exactly implemented functional and non-functional requirements of the software product will be. This may for example lead to a dissatisfied customer and to unforeseen, even potentially dangerous situations. Apart from the fact that people's lives may be at risk, such situations can have an enormous financial impact on the supplier as the operational maintenance cost is also unknown. The post-release or maintenance cost of the software products may be unexpectedly high. The exact status of the software product with its documentation is not known, which may cause high corrective maintenance cost when repairing failures. Further, adaptive and perfective maintenance activities may be severely hampered.
- ***Weak foundation for selecting improvements***. Not analysing the reported failures after product release possibly prevents an organisation from selecting the best improvements in the development process. It may for instance be the case, that the measured productivity during development should be compensated for a very high level of corrective maintenance cost due to many reported failures. In that case, an improvement of the reliability by increasing inspections might be a far better improvement than for instance documenting policies as a prerequisite to reach the next CMM level.

This sub optimisation is often a result of separating the responsibilities of product development and product maintenance. A (project) manager responsible for product development only, will for instance be less willing to invest in measures to improve the maintainability of a product. It means spending additional budget and time, where the assigned responsibility is often to limit both budget and lead-time in order to release the product as early as possible against minimal cost.

In the next section, it will be investigated how the definition of PSP metrics can be promoted to the level of a project and what extensions must be made to cover the entire lifecycle of a software product instead of focusing on the development phase only. By doing so, a measurement foundation is built that offers management a powerful instrument to support quantitative decision-making throughout the lifecycle of a software product.

6. Proposed Lifecycle Model SLCM

The PSP metrics can be easily promoted to the level of a project. By adding some additional metrics, a lifecycle model is obtained which enables management to evaluate projects with regard to predictability (estimation), effectiveness (quality) and efficiency (productivity). Analysis of the data and discussing the results has two main advantages. In the first place, the possibility is offered to remove process deficiencies, which are nothing less than process improvements. Secondly, calculations can be made of the expected return on investment. These issues will be discussed in section 8. In this section and section 7, the lifecycle model and derived metrics will be discussed first.

In Figure 4 the proposed lifecycle model ***SLCM*** (***S****oftware* ***L****ife****C****ycle* ***M****odel*) is presented together with the derived metrics. In this section, the data to be supplied in the model will be discussed. The derived metrics are the subject of section 7.

<Figure 4 to be inserted here>

The data to be supplied to the model is the following:

- ***Size***. Both the estimated size of the product to be developed and the actual size of the developed product are inputs to the model. The estimated size will be obtained at the beginning of the project; the actual size when the product is ready to be released. Size can for instance be expressed in Kilo Lines Of Code (KLOC), Function Points (FP), or other units.
- ***Effort***. Regarding effort, distinction is made between different phases and different categories. The phases can be tailored to the ones used in a specific organisation. The same holds for the categories, however it is important to make an explicit distinction between the effort spent on

appraisal activities (like inspections and reviews), failure activities (rework for detected defects), and other effort (in this case management and technical work). For each category in each phase, the estimated effort to be spent and actual spent effort are inputs.

- ***Time***. Distinction is made between the estimated lead-time and actual elapsed lead-time in days per phase. The fact that phases may overlap each other is not taken into account here.
- ***Defects***. Finally, the detected and repaired defects are input to the model. For each phase, the detected and repaired defects must be counted and it must be determined in which phase each defect was injected.

This data highly corresponds with the data an individual software engineer collects when applying the PSP. The main differences are that now the maintenance phase is also taken into account and that the total effort spent in each phase is distributed over different categories.

7. Derived Metrics

Being able to collect the data as input to the lifecycle model ***SLCM*** offers a project the possibility to derive powerful metrics that support decision-making in future projects and to select improvements with a possibly high return on investment. These metrics are presented in the lower part of Figure 4 and are now shortly described:

- ***Estimation Accuracy***. The estimates with regard to size, effort and time can be compared with the actual figures when the product is ready to be released and during further maintenance activities. The result is the estimation inaccuracy, which can be computed with equation ***(1)***. In Figures 5a and 5b, the effort and time estimations are graphically compared with the effort and time actuals respectively. It also reveals how effort and time are distributed over the different phases.

- ***Effort Distribution per phase***. Another important metric is the distribution of the different effort categories over the various phases. See Figure 5c. This gives insight in important issues like for instance the appraisal and failure cost in each phase.

- ***Yield***. Regarding the Yield, distinction is made between the Yield of phase X and the cumulative Yield of phase X and all its previous phases. See Figure 5d. The following equations are used:

$$Yield\ (X) = \frac{\#\ defects\ removed\ in\ phase\ X}{(\#\ defects\ from\ phase\ X\text{-}1) + (\#\ defects\ injected\ in\ phase\ X)} \qquad (10)$$

$$Yield\ (X)_{cumulative} = \frac{\#\ defects\ removed\ so\ far}{\#\ defects\ injected\ so\ far} \qquad (11)$$

- ***Appraisal/Failure Ratio***. The A/FR is also calculated for each phase X separately as well as for the combination of phase X and all its previous phases (appraisal cost) compared to all its subsequent phase (failure cost). See Figure 5e. The following equations are used:

$$Appraisal/Failure\ Ratio\ (X) = \frac{appraisal\ effort\ in\ phase\ X}{failure\ effort\ in\ phase\ X} \qquad (12)$$

$$Appraisal/Failure\ Ratio\ (X)_{cumulative} = \frac{cumulative\ appraisal\ effort\ so\ far}{cumulative\ failure\ effort\ in\ subsequent\ phases} \qquad (13)$$

- ***Defect Detection Rate and Defect Removal Rate***. The Defect Detection Rate and Defect Removal Rate are metrics indicating how much effort is needed to detect a defect and how much effort is needed to remove a defect. They are directly related to the appraisal cost and failure cost respectively. See also Figures 5f and 5g. The equations are:

$$Defect\ Detection\ Rate\ (X) = \frac{defects\ detected\ in\ phase\ X}{appraisal\ effort\ in\ phase\ X} \qquad (14)$$

$$\text{Defect Removal Rate (X)} = \frac{\text{defects removed in phase X}}{\text{failure effort in phase X}} \quad (15)$$

- ***Productivity***. Distinction is made between the productivity during development and the productivity taking into account the maintenance phase as well. Equations:

$$\text{Productivity}_{development} = \frac{\text{actual product size}}{\text{development effort}} \quad (16)$$

$$\text{Productivity}_{overall} = \frac{\text{actual product size}}{\text{development effort + corrective maintenance effort}} \quad (17)$$

- ***Measured Reliability***. The reliability is measured continuously during the maintenance phase and recalculated when a new failure has been reported and accepted. Equation:

$$\text{Reliability}_{measured} = \frac{\text{reported failures}}{\text{productsize in KLOC}} \quad (18)$$

- ***Technical Metrics***. Finally, two technical metrics are calculated and can be of help with respect to the maintenance phase, especially when subsequent versions of the product are being developed. They cannot be calculated from the supplied input data in the model, but must be calculated separately. The first one is the Maintainability Index or MI [OMA 1994] and gives an indication how maintainable a software product is. [7] There are two equations available, the second one taken into account the availability of comment in the code (assuming that it has a positive influence on maintainability):

$$MI = 171 - 3.42\ ln(aveV) - 0.23\ aveV(g') - 16.2\ ln(aveLOC) \quad (19)$$

$$MI = 171 - 3.42\ ln(aveV) - 0.23\ aveV(g') - 16.2\ ln(aveLOC) + 50\ sin\sqrt{2.46\ perCM} \quad (20)$$

with:

aveV	=	*average Halstead Volume per module*
aveV(g')	=	*average extended cyclomatic complexity per module*
aveLOC	=	*average lines of code per module*
perCM	=	*average percent of lines of comment per module*

The second technical metric is the Software Maturity Index or SMI [IEE 1988a, IEE 1988b].

$$SMI = [\ M_t - (\ F_a + F_c + F_d)\ ]\ /\ M_t \quad (21)$$

with:

M_t	=	*number of modules in the current release*
F_c	=	*number of modules in the current release that have been changed*
F_a	=	*number of modules in the current release that have been added*
F_d	=	*number of modules in the current release that have been deleted*

SMI provides an indication of the stability of a software product and can be used as a metric for planning software maintenance activities. As SMI approaches 1, the product begins to stabilise.

<Figure 5 to be inserted here>

[7] *MI < 65: poor maintainability, 65 <= MI < 85: fair maintainability, MI >= 85: excellent maintainability [COL 1993].*

8. Using SLCM

This set of derived metrics offers the possibility to analyse projects and their products into detail. Take a look for instance at the example as presented in Figure 4 and Figure 5. The following can be concluded:

- ***Predictability***. There is a considerable budget overrun (+50%) and schedule overrun (+19%), but the realised product size is less than predicted (-24%). The realised overall productivity is nearly half the predicted value (-49%).[8] These data need to be further analysed in order to find out the reasons for these deviations.
- ***Effectiveness***. The cumulative Yield at the Detailed Design phase and the Implementation phase is very low, 10% and 19% respectively. This implies that many defects are transported to subsequent phases and that the failure cost in these phases will probably be high. This is confirmed by the fact that the total amount of rework equals 38% of the overall lifecycle effort.
- ***Efficiency***. The efficiency is very low because many defects have been found in the later phases. As a consequence, the overall failure cost is high (38% of overall lifecycle effort). This is confirmed by the fact that the cumulative A/FR before the Detailed Design phase and Implementation phase is very low (0.02 and 0.03 respectively).

This analysis gives an organisation the possibility to select improvements for future projects, assuming that this project has not been very exceptional but comparable with similar projects. Improvements might be:

- Improve the predictability by using a lower productivity figure when estimating projects. In this way, more realistic estimates will be obtained.
- Increase the quality of the resulting product by increasing the Yield in the earlier development phases. This can be done by increasing the review effort (A/FR higher) or by adjusting the review method.
- Improve the efficiency by increasing the A/FR in the various development phases before the Integration Test phase. Special interest might be given to the Implementation phase and Unit Test phase as in these phases the Detection Rate and Removal Rate are very high.

In all cases, the available data can be used to make predictions about the expected effects in terms of return on investment. This is not always easy, as the various parameters might be correlated (for instance, a higher A/FR will probably contribute to a higher Yield). However by building a record of experiences, the capabilities in this area can be stepwise improved. In this example the following approach might be applied in an iterative way:

1. Increase the cumulative A/FR by increasing the appraisal cost in the earlier development phases.
2. Estimate what the effect will be on the cumulative Yield at the Detailed Design phase: what will be the reduction of defects injected in the Requirements phase and Architecture phase?
3. Take measures to improve the inspection and review process during the Implementation phase and Unit Test phase and estimate the effect on the Defect Detection Rate and the Defect Removal Rate in these phases.
4. Use the cumulative Yield at the Detailed Design phase to predict the number of defects that will be passed forward to subsequent phases and use the Defect Detection Rate and Defect Removal Rate to calculate the failure cost.
5. Calculate the effect on effort, time, and expected corrective maintenance cost.
6. Repeat the steps 1 till 5 until satisfactory (and realistic) values have been obtained.

The presented model ***SLCM*** and derived metrics is considered to be useful for organisations still having to build a measurement foundation for their software projects and products as well as for organisations looking for further improvements of their software processes. The mature organisations will analyse their available data resulting from ***SLCM***, select the most promising improvements and re-measure their performance over time. The less mature organisations will not be able to implement

[8] *Note here, that many organisations tend to calculate the productivity using the development effort only, see equation (16). It is advised however to use equation (17), giving a far more realistic view.*

SLCM from day one, but need to define a strategy how to implement the model in subsequent steps. The order of the steps can be derived from the most urgent problems the organisation is faced with:

- ***Predictability***. In this case, the first step might be to collect estimates and actuals for size, effort and time.
- ***Effectiveness***. In this case, the focus is on the output of the project, being the quality of the product. Important measurements to be considered first are defects (Yield values per phase and cumulative).
- ***Efficiency***. In this case, attention must be paid to appraisal cost related to failure cost (A/FR) and defect rates (Defect Detection Rate and Defect Removal Rate).

In all cases, supporting methods and tools might be selected to facilitate the measurements, as long as it is understood that *"a fool with a tool remains a fool"*. A method or tool will not solve the problem; they can only support a defined process.

The presented model ***SLCM*** is simple and easy to understand, which is helpful to become adopted by organisations. The final commitment to its adoption will however probably need tailoring the model to the characteristics of an organisation, its way of working and its products. There may be a need to change metrics definitions, leave some metrics out and add other ones. On a lower level, more detailed information may be generated enabling in-depth analysis. An example is for instance using Orthogonal Defect Classification to analyse defect data [CHI 1992]. Beyond this, as any other model ***SLCM*** also has its limitations. The most important limitation might be that it focuses on the initial development of a new product and regards all subsequent versions being part of the maintenance phase. This limitation might be overcome in the following way:

- A separate model is used for each product version.
- When performing corrective maintenance activities each failure and related effort are allocated to the product version in which the corresponding defect was introduced.

9. Conclusions and Next Steps

In this article, the software lifecycle model ***SLCM*** was introduced. The model offers a measurement foundation for software projects and products, enabling an organisation to analyse the predictability (estimation capabilities), effectiveness (product quality) and efficiency (process productivity). The model is based on PSP metrics, which have been promoted to the level of a project and definitions have been slightly adjusted to include the maintenance phase as well. This approach might be of help to any organisation irrespective of their maturity regarding software process and metrics. The strength of this simple approach is that every professional manager will understand the need to collect this data in order to support quantitative decision-making. Its simplicity might be a weakness, however it offers two advantages. In the first place, a decision-maker simplifies reality and prefers simple rules of thumb as a consequence of limited cognitive capabilities [SIM 1982]. The model supports this concept of bounded rationality. Overly complex models will be adopted less easily. In the second place, the model focuses on macroscopic measurements. Getting data at this level and being able to make the right interpretations is the first step in maturing the software industry. Only when these improvements have proved to be successful and sustainable, the granularity of measurements can be enlarged.

At this moment, the concept version of ***SLCM*** has been distributed to a limited number of organisations. They have shown their interest in using the model and will give feedback of their findings and results. This must lead to an improved and official version 1.0 of the model. Organisations willing to participate and use the materials are kindly requested to contact the author.

About the author

Hans Sassenburg (*hsassenburg@se-cure.ch*) received a Master of Science degree in Electrical Engineering from the Eindhoven University of Technology in 1986 (The Netherlands). During his study years he founded a one-person consulting firm and worked as an independent consultant. In 1996 he co-founded a new consulting and training firm (Alert Automation Services b.v.). From 1996 till 2001 he worked as a guest lecturer and assistant professor at the Eindhoven University of Technology. In 2001 he moved to Switzerland where he founded a consulting firm (SE-CURE AG, *www.se-cure.ch*). Having finished his first book *"Software Engineering: van ambacht naar professie"* (ISBN 90-72194-64-0), he started a Ph.D. research at the Faculty of Economics at the University of Groningen

(The Netherlands) under the supervision of Prof. Egon Berghout (e.w.berghout@eco.rug.nl). His second book *"Software Development from an Economic Perspective"* will be published in 2004.

References

[BER 2001] *"Full life-cycle management and the IT Management paradox"*, E.W.Berghout, M.Nijland, in D.Remeny & A.Brown (Eds.): "Make or Break Issues in IT Management", Butterworth-Heinemann.

[BOE 1981] *"Software Engineering Economics"*, B.W.Boehm, Prentice-Hall.

[CHI 1992] *"Orthogonal Defect Classification – A Concept for In-Process Measurements"* R.Chillarege et al, IEEE Transactions on Software Engineering, Vol.18, No.11.

[COL 1993] *"The Application of Software Maintainability Models on Industrial Software Systems"*, D.Coleman et al., University of Idaho, Software Engineering Test Lab, Report No. 93-03 TR.

[HUM 1995] *"A Discipline for Software Engineering"*, W.S.Humphrey, Addison-Wesley.

[HUM 1997] *"Introduction to the Personal Software Process"*, W.S.Humphrey, Addison-Wesley.

[IEE 1988a] *"IEEE Standard Dictionary of Measures to Produce Reliable Software"*, IEEE Std. 982.1, The Institute of Electrical and Electronics Engineers.

[IEE 1988b] *"IEEE Guide for the Use of IEEE Standard Dictionary of Measures to Produce Reliable Software"*, IEEE Std. 982.1, The Institute of Electrical and Electronics Engineers.

[IEE 1998] *"IEEE Standard for Software Maintenance"*, IEEE Std. 1219, The Institute of Electrical and Electronics Engineers.

[KIT 1996] *"Software Quality: The Elusive Target"*, B.Kitchenham, S.L.Pfleeger, IEEE Software, Vol. 13, No. 1, pp. 12-21.

[McK 1984] *"Maintenance as a function of design"*, J.R.McKee, Proceedings AFIPS National Computer Conference, Las Vegas (USA).

[NOS 1990] *"Software Maintenance Management: changes in the last decade"*, J.T.Nosek, P.Palvia, Software Maintenance: Research and Practice (2).

[OMA 1994] *"Constructing and testing of Polynomials Predicting Software Maintainability"*, P.Oman, J.Hagemeister, Journal of Systems and Software, Vol. 24 (3), March.

[SAS 2002] *"Reviews: why and how?"*, J.A.Sassenburg, Informatie, October, pp. 16-21 (in Dutch).

[SAS 2003] *"When to release the software?"*, J.A.Sassenburg, Proceedings of the European SEPG, London (UK).

[SEI 2002] *"Process Maturity Profile of the Software Community, 2001 Year End Update"*, website www.sei.cmu.edu, Software Engineering Institute.

[SIM 1982] *"Models of Bounded Rationality"*, H.Simon, Vol. 2, Cambridge, MIT Press.

[ZAH 1998] *"Software Process Improvement - Practical Guidelines for Business Success"*, S.Zahran, Essex: Addison Wesley Longman.

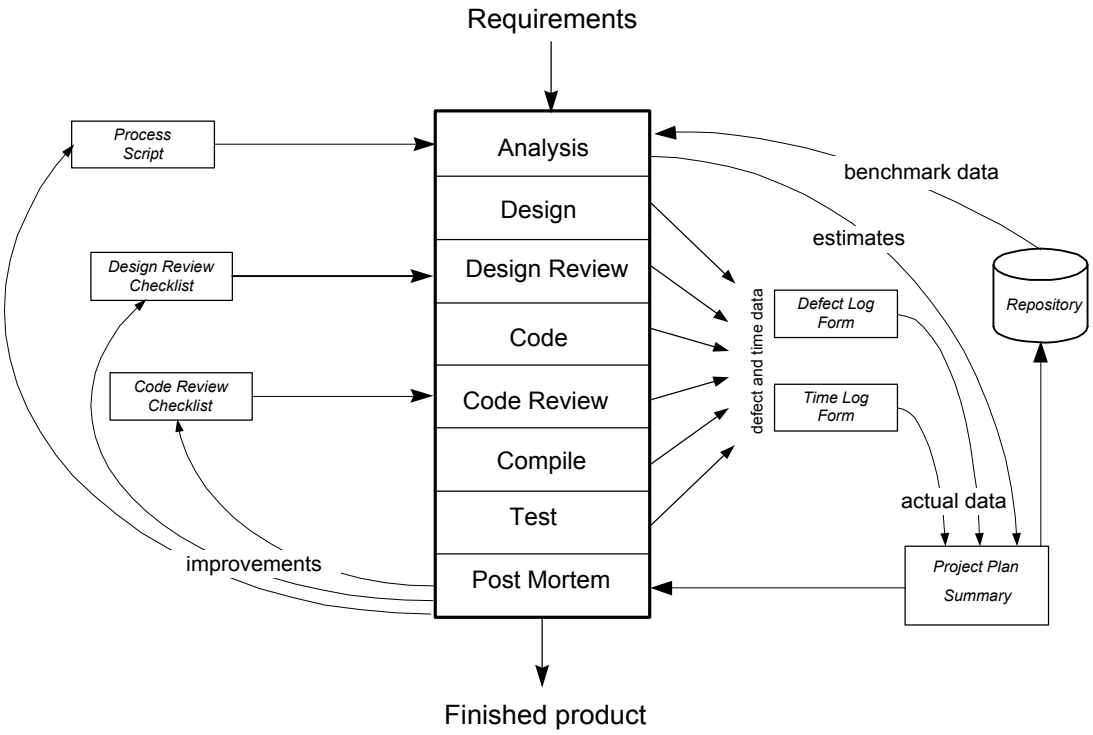

Figure 1: Process flow of the Personal Software Process (based on [HUM 1997, p. 125]).

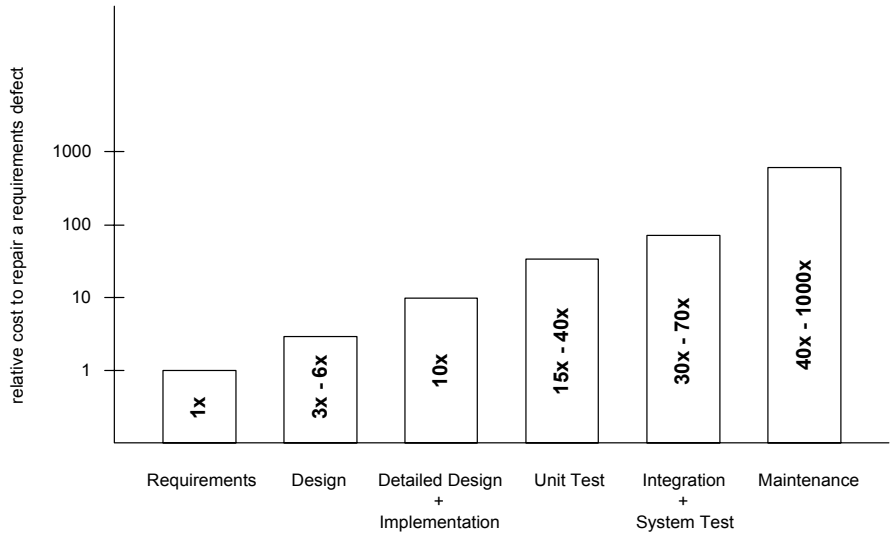

Figure 2: Repair cost of a requirements defect [BOE 1981].

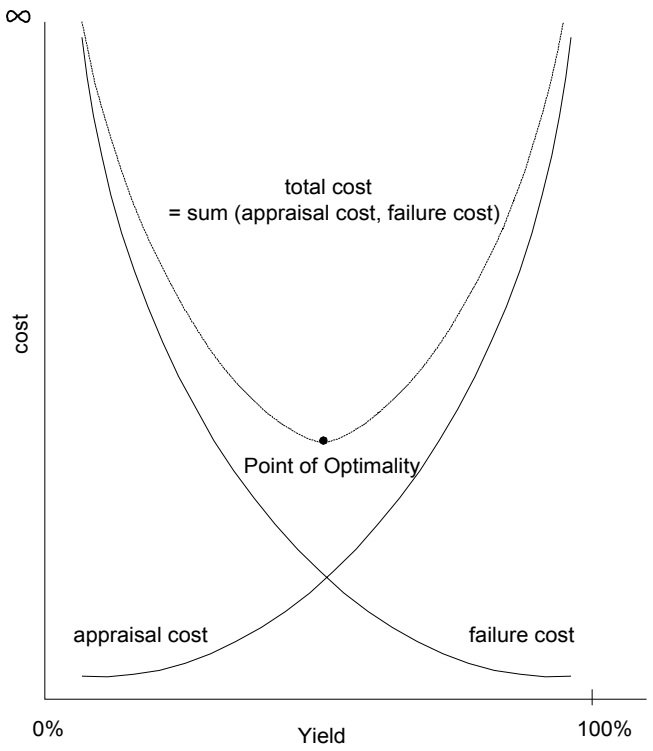

Figure 3: Relationship between appraisal cost, failure cost, and Yield [SAS 2002]. [9]

[9] *Figure 2 is just an example. The slope of the curves and thus also their intersection depend on factors like the type of application and the maturity of the development process. This also means that the minimum of the sum of preventive cost and rework cost is not necessarily the intersection of the two curves (A/FR equal to 1).*

SLCM (version 0.1)	**Size (KLOC)**		**Effort (Hours)**								**Time (Days)**		**Defects (count)**							
			Management		Technical		Appraisal		Failure											
	Est.	Act.	Est.	Act.	Est.	Act.	Est.	Act.	Est.	Act.	Est.	Act.	R	A	DD	I	UT	IT	ST	M
Requirements			100	110	200	240	50	16	20	8	40	30	1							
Architecture			150	170	450	500	50	16	40	10	50	40	2	3						
Detailed Design			200	230	800	500	70	60	60	50	80	60	2	2	9					
Implementation			200	260	800	1000	70	40	60	50	120	150	4	4	24	15				
Unit Test			200	300	800	1200	30	20	60	50	30	60	6	3	20	15	5			
Integration Test			200	270	800	400	0	250	40	400	20	70	12	7	13	18	5	4		
System Test			200	270	600	200	0	350	30	800	10	80	18	6	7	4	5	6	7	
Maintenance			50	120	0	0	0	0	700	2650	400	400	9	14	27	103	5	9	4	8
Total	100	76	1300	1730	4450	4040	270	752	1010	4018	750	890	54	39	100	155	20	19	11	8

Maintainability Index	60
Software Maturity Index	0.3

Estimation		R	A	DD	I	UT	IT	ST	M	Total
	Estimated	370	690	1130	1130	1090	1040	830	750	7030
	Actual	374	696	840	1350	1570	1320	1620	2770	10540
	Effort Diff.	1%	1%	-26%	19%	44%	27%	95%	269%	**50%**
	Estimated	40	50	80	120	30	20	10	400	750
	Actual	30	40	60	150	60	70	80	400	890
	Time Diff.	-25%	-20%	-25%	25%	100%	250%	700%	0%	**19%**

Summary	Est.	Act.	Diff.
Size	100	76	-24%
Effort	7030	10540	50%
Time	750	890	19%
Productivity	14.22	7.21	-49%

Distribution	**Effort**	R	A	DD	I	UT	IT	ST	M	Total
	Management	1%	2%	2%	2%	3%	3%	3%	1%	16%
	Technical	2%	5%	5%	9%	11%	4%	2%	0%	38%
	Prevention	0%	0%	1%	0%	0%	2%	3%	0%	7%
	Rework	0%	0%	0%	0%	0%	4%	8%	25%	38%
	Total	4%	7%	8%	13%	15%	13%	15%	26%	100%

Time	R	A	DD	I	UT	IT	ST	M
Phase	3%	4%	7%	17%	7%	8%	9%	45%
Cumulative	3%	8%	15%	31%	38%	46%	55%	100%

Process	**Yield**	R	A	DD	I	UT	IT	ST	M
	Phase	2%	5%	7%	14%	16%	22%	24%	100%
	Cumulative	2%	6%	10%	19%	31%	45%	57%	100%

A/FR	R	A	DD	I	UT	IT	ST	M
Phase	2.00	1.60	1.20	0.80	0.40	0.63	0.44	0.00
Cumulative	0.00	0.01	0.02	0.03	0.04	0.12	0.28	-

Others	R	A	DD	I	UT	IT	ST	M	
Detection	0.06	0.31	0.22	1.18	2.45	0.24	0.15	0.00	defects/hour
Removal	0.13	0.50	0.26	0.94	0.98	0.15	0.07	0.07	defects/hour

Productivity R - ST	9.78	LOC/hour	
Productivity R - M	7.21	LOC/hour	
Measured Reliability	2.36	failures/KLOC	

Figure 4: SLCM datasheet.

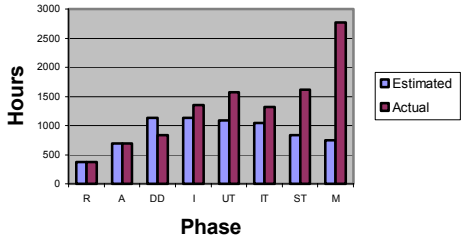

Figure 5a: Effort Estimation Accuracy.

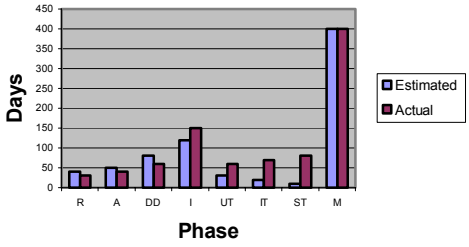

Figure 5b: Time Estimation Accuracy.

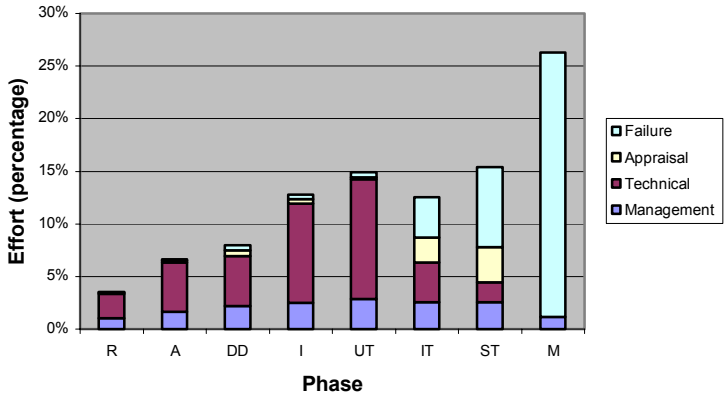

Figure 5c: Categorized Effort per Phase.

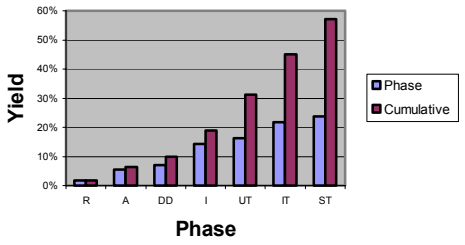

Figure 5d: Yield.

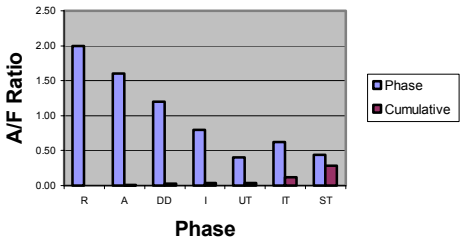

Figure 5e: Appraisal/Failure Ratio.

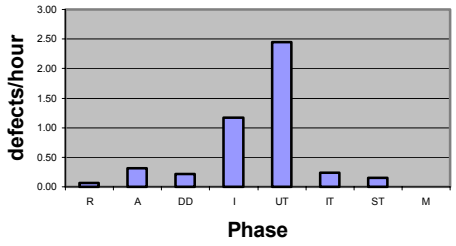

Figure 5f: Defect Detection Rate.

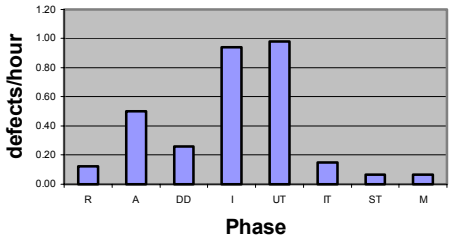

Figure 5g: Defect Removal Rate.

Figure 5: SLCM Graphs.