SOFTWARE RELIABILITY ASSESSMENT

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

DENİZ KAYA

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
ELECTRICAL AND ELECTRONICS ENGINEERING

AUGUST 2005

Approval of the Graduate School of Natural and Applied Sciences

____

Prof. Dr. Canan Özgen
Director

I certify that this thesis satisfies all the requirements as a thesis for the degree of Master of Science.

____

Prof. Dr. İsmet Erkmen
Head of Department

This is to certify that we have read this thesis and that in our opinion it is fully adequate, in scope and quality, as a thesis for the degree of Master of Science.

____

Prof. Dr. Semih Bilgen
Supervisor

Examining Committee Members

Assist. Prof. Dr. Cüneyt Bazlamaçcı	(METU,EE)	________________________
Dr. İlkay Ulusoy	(METU,EE)	________________________
Dr. Şenan Ece Schmidt	(METU,EE)	________________________
Prof. Dr. Semih Bilgen	(METU,EE)	________________________
Hüseyin Türkoğlu	(TÜBİTAK-SAGE)	________________________

ABSTRACT

SOFTWARE RELIABILITY ASSESSMENT

Kaya, Deniz
M.S., Department of Electrical and Electronics Engineering
Supervisor: Prof. Dr. Semih BİLGEN

August 2005, 117 Pages

In spite of the fact that software reliability studies have attracted great deal of attention from different disciplines in 1970s, applications of the subject have rarely been involved in the software industry. With the rise of technological advances especially in the military electronics field, reliability of software systems gained importance.

In this study, a company in the defense industries is inspected for their abilities and needs regarding software reliability, and an improvement proposal with metrics measurement system is formed. A computer tool is developed for the evaluation of the performance of the improvement proposal. Results obtained via this tool indicate improved abilities in the development of reliable software products.

Keywords: Software Reliability, Software Process Improvement, Software Process Simulation

ÖZ

YAZILIM GÜVENİLİRLİĞİ DEĞERLENDİRİMİ

Kaya, Deniz
Yüksek Lisans, Elektrik-Elektronik Mühendisliği Bölümü
Tez Yöneticisi: Prof. Dr. Semih BİLGEN

Ağustos 2005, 117 Sayfa

Yazılım güvenilirliği alanındaki çalışmaların 1970'li yıllarda farklı disiplinlerden önemli ölçüde ilgi görmüş olmasına rağmen, konuya ilişkin uygulamalardan yazılım sanayiinde nadiren yararlanılmıştır. Özellikle askeri elektronik alanındaki teknolojik ilerlemelerin yükselişiyle birlikte, yazılım sistemlerinin güvenilirliği önem kazanmıştır.

Bu çalışmada, savunma sanayiinde faaliyet gösteren bir kuruluşun yazılım güvenilirliğine ilişkin yetenekleri ve gereksinimleri incelenmiş ve metrik ölçüm sistemi içeren bir iyileştirme önerisi oluşturulmuştur. İyileştirme önerisinin başarımının değerlendirilmesi için bilgisayar ortamında çalışan bir araç geliştirilmiştir. Bu araçla elde edilen sonuçlar, güvenilir yazılım ürünlerinin geliştirilmesine yönelik yeteneklerin iyileştirildiğini göstermektedir.

Anahtar Kelimeler: Yazılım Güvenilirliği, Yazımım Süreci İyileştirme, Yazılım Süreci Benzetimi

To the memory of my father …

ACKNOWLEDGMENTS

The author would like to express his feelings of gratitude to his supervisor Prof. Dr. Semih Bilgen for his skillful guidance, endless patience, encouragement and support throughout this study.

Contributions and cooperation of TÜBİTAK-SAGE administration and employees are gratefully acknowledged.

The author would also like to thank his dear wife, Nihal, for her never-ending morale support and extreme patience.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

CD	Compact Disc
COCOMO	Constructive Cost Model
GQM	Goal Question Metrics
ID	Identity
IEEE	The Institute of Electrical and Electronics Engineers, Inc
IOR	Input, Output, Result
KLOC	Kilo LOC
KLOCE	KLOC Estimate
LOC	Lines of Code
MDD	Mean Determination Date
MFD	Mean Formation Date
MRT	Mean Removal Time
MTbFD	Mean Time between Formation and Determination
MTbFR	Mean Time between Formation and Removal
NHPP	Non-Homogeneous Poisson Process
NoFD	Number of Faults Determined
NoFI	Number of Faults Introduced
RODF	Ratio of Determined Faults
SRWG	Software Reliability Work Group
TTE	Total Time Elapsed
YUB	Yazılım Uygunsuzluk Bildirim Formu (Software Error Notification Form)

CHAPTER 1

INTRODUCTION

Software reliability denotes the probability that software in a pre-defined condition performs its tasks without malfunctioning for a specified duration. It may be regarded as a component of software quality. Unlike software quality, however, it concentrates on the functionality of the software and disregards such issues as ergonomics of software products, development economics, etc. unless they constitute functional attributes of the software product.

In order to express the reliability of a software product quantitatively, first, the product itself must be “measured”. For this purpose, the abstraction of measurement has to be removed. This can be achieved by defining certain measures, or metrics, about software product and its development process.

Once reliability metrics are defined, it is wise to question if it is possible to determine and improve the reliability of software with a system based on these metrics.

In this work, the problem of measurement and improvement of reliability of software products developed at a company shall be investigated. For this purpose, first of all, previous studies on the field of software reliability are investigated. As a next step, the costs and benefits associated with collecting reliability metrics in the specific company are investigated. Then, a set of metrics are selected for the purpose of enabling the Company to construct a measurement system. The next task is the development of a proposal for improvement of software development processes of the Company. This study does not aim to propose an improvement to the software development processes in general sense; rather it introduces minor modifications to existing processes with software reliability being the primary concern. Once the proposal is formed, different evaluation alternatives are presented. While a real-life improvement project would be definitely more realistic, in this study, a simulation-based evaluation is performed. Finally, obtained results are questioned if proposed system meets the needs of the Company, and if proposed actions result in expected improvements to software reliability.

Chapter 2 of this study contains a survey on the software reliability literature. General concepts of software reliability and their application areas are discussed in that chapter.

In Chapter 3, software development system of the Company is inspected and a system for measurement of software reliability is developed.

Alternatives for evaluation of the proposed system and the tool developed for simulation-based evaluation is discussed in Chapter 4.

Chapter 5 is devoted to the discussion of results generated by using the tool discussed in Chapter 4.

Chapter 6 presents conclusions regarding this study and suggestions for future work.

CHAPTER 2

LITERATURE SURVEY

In this chapter, first the terminology to be used in the rest of this study is presented. Then, studies on software reliability are summarized in subsections according to their relevance to different aspects of software reliability: Assessment of Software Reliability, Quality, and Project Management. Finally, general characteristics of metric collection systems, which also constitute a major task in this study, are presented.

2.1 Definitions

For the sake of consistency, all of the definitions are directly taken from [1].

Defect: A product anomaly. Examples include such things as (1) omissions and imperfections found during early life cycle phases and (2) faults contained in software sufficiently mature for test or operation.

Fault: (1) An accidental condition that causes a functional unit to fail to perform its required function. (2) A manifestation of an error in software. A fault, if encountered, may cause a failure. It is synonymous with 'bug'.

Failure: (1) The termination of the ability of a functional unit to perform its required function. (2) An event in which a system or system component does not perform a required function within specified limits. A failure may be produced when a fault is encountered.

Error: Human action that results in software containing a fault. Examples include omission of misinterpretation of user requirements in a software specification, incorrect translation, or omission of a requirement in the design specification.

Measure: A quantitative assessment of the degree to which a software product or process possesses a given attribute.

Software Reliability: The probability that software will not cause the failure of a system for a specified time under specified conditions. The probability is a function of the inputs to and use of the system as well as a function of the existence of faults in the software. The inputs to the system determine whether existing faults, if any, are encountered.

2.2 Studies on Software Reliability

The IEEE defines software reliability as the probability that software will not cause the failure of a system for a specified time under specified conditions [1]. While the definition is rather simple, its implications constitute a wide research field with sub-fields of different concerns. These sub-fields can be investigated in three major classes:

- Assessment of Software Reliability
- Quality
- Project Management

In spite of the fact that above classes have close interaction, it is necessary to make such a classification for the sake of ensuring extensive study of the subject. In the rest of this section, sub-fields of software reliability are studied.

2.2.1 Assessment of Software Reliability

Apart from classical hardware reliability, software reliability has rather different nature [2, 3, 23]. While the reliability of hardware continues to change even after the product is delivered, the reliability of software is improved throughout the development process until the product is delivered. This matter is pointed out in Figure 1 [2]. After the delivery, a change in reliability level is possible only if maintenance action is performed to either compensate for defects in the software or to catch up with technological advances.

Another major difference between software reliability and hardware reliability is that software reliability is not a function of how frequent that specific software is used, whereas hardware is subject to wear out [23, 26]. Also, because software is rather conceptual, documentation is considered as an integral part of software and software reliability [3].

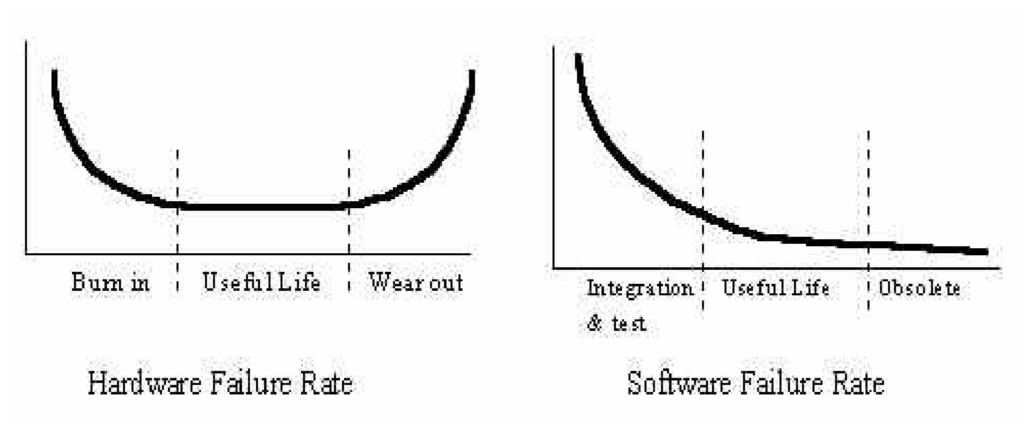


Figure 1 - Comparison of hardware and software reliability growth.

A common constituent of hardware and software reliability techniques is testing [4]. The results of testing process are employed in software reliability growth models to translate defect and/or failure data into reliability measures [24, 25].

Because of all these common points and differences mentioned, it is wise to classify studies on assessment of software reliability into two groups: Software Reliability Modeling, and Software Testing.

2.2.1.1 Software Reliability Modeling

In prediction and estimation of software reliability a general method is the use of statistical models [25]. These models make use of either historical data of similar projects or organizations or direct software measures such as fault density, defect density, and defect detection rate of the software under investigation [1, 3, 24, 25].

Some of the well-known examples of software reliability models are Musa's Execution Time Model [2], Putnam's Model [3], Goel-Okumoto Model [28], Generalized Goel NHPP Model [5], Jelinski-Moranda Model [11], and Littlewood-Verrall Model [14]. All these models, as expected, have their own set of advantages and disadvantages that take their roots from their specific assumptions [11].

In addition to those model approaches, there exist other techniques for assessment of software reliability [12]. Test coverage techniques, execution path and error seeding are examples of these alternative approaches [13].

In the literature different approaches to estimation of the reliability of a software program have been reported [5, 6, 7]. The problem with the estimation approach is that it can only be used at later stages of software development process, which channels organizations to use of reliability prediction techniques [6].

Software reliability prediction techniques are especially useful when knowledge of approximate reliability level of the software to be developed is desired at early stages of development life cycle [3]. When that information is of critical importance, the performance of prediction process in determination of an initial guess can be improved by the use of more than one prediction model over the same data [8, 27].

One of the major problems of software reliability prediction models is that they fail to predict the reliability accurately [9]. The reason is that they assume limited historical data of special kind of organizations or of specific type of projects [3]. That creates the problem of loss of control over customization of model's criteria to fit it to a specific organization [9]. Reliability estimation models can overcome this problem up to some extent [10].

The estimation models are usually in the form of non-homogeneous Poisson processes (NHPP) or Markoff systems [11]. Most of the time the difference between the models arises from the definition (or assumption) of "beginning time of the process" or selection of random variable of the model as being either "number of faults detected" or "total number of faults predicted" [3]. In the literature, however, it is possible to come across with models that do not require detection of all the failures [15]. Models that relate reliability to cost and priority of failures also exists [16].

2.2.1.2 Software Testing

Software reliability efforts and software testing process complements each other: The results of software testing provide statistical data to model the reliability, and the reliability level of the software determines the amount of necessary testing [14, 17].

In order to provide reliability assessment process with healthy input data, the testing of software must be comprehensive and complete both in terms of user requirements and software architecture [3, 23]. While well-known software engineering sources [21] suggest ways to improve testing process, reliability-oriented studies are still worth mentioning.

The major difference between the viewpoints of "software engineers" and of "software reliability engineers" is that the former is mostly interested in the coverage of functionalities and flow paths, whereas the latter is interested in coverage of failures (or defects) [13, 22].

There, however, exist some problems with software testing process when software reliability is of primary concern. The first problem with software testing is coverage: Because of the direct effect of the selection of failure data on the reliability model performance, the content and coverage of the tests are critical [9, 18]. Coverage problem also affects the cost of a project since the cost of finding a defect in early phases of software development process is lower than that of finding it later in the development process [16]. Another important aspect of test coverage is that the selection of test cases and failure data influences the way the software reliability estimation model are formed [9].

The second problem is detection and prevention of failures; not every failure is an independent one and it is possible that removal of a failure also remove (or introduce) another one. That is why nature of the failures should be investigated to see if there is correlation between failures [19]. At this point, the study of Wohlin and Korner gains importance [20]. In that study a model has been formed to represent the spread of defects based on a level-approach, in which the term "level" corresponds to the phase of the development process that a specific fault is first introduced. It is stated in that study that a defect found in a level can be the indicator of the defects in previous levels.

In contrast to coverage of functionality, which is some sort of validation of what is intended to implement, the business of failure coverage is not a straight-forward action due to stochastic nature of distribution of failures. Wohlin and Korner's method solves this problem up to some extent [20]. However their assumption that a failure in a level is independent of the others cause problem in real-life [19]. In deed, the relation of a defect found in early phases of the project with another one found in later steps is not covered in their study.

An idea to relax the testing process, which is proposed by Boland and Singh [16] is that the effect of finding an error in early phases has more noticeable effect on the overall failure rate of the software than that of finding it later. That idea leads to the corollary that it is helpful to spend more effort on testing at early stages, beginning at component testing and code-review.

There are some studies to determine a method to guarantee failure coverage. Some researches prefer use of test-coverage methods to defect-coverage and generate the concept of test-coverage growth [13].

It is proven in another study that ability to detect defects is correlated with code-coverage [22]. A method is formed in that study for this purpose and the results are compared with well-known software reliability growth models to determine their accuracy.

2.2.2 Quality and Software Reliability

Software reliability is considered as an important metric for software quality [1, 3, 18, 26]. In [29], however, Voas indicates that highly-reliable software is not necessarily a high-quality product, as there exist situations in which ultra-reliable software systems showed performance degradations, poor robustness and lack of maintenance precautions.

An approach proposed to make reliability estimations and predictions parallel to quality is to organize the testing process in such a way to make the user requirements tested more strictly with increased frequency of repetition of revealing input set [2, 30]. The essence of this technique is that most of the time the user is not interested in how the problem was solved; he/she wants to see that the proposed solution is the one that meets the requirements.

The problem with the method mentioned above is that exception handling is not always considered when such testing scenarios are created [31]. Especially in the case of safety-critical software, it is difficult to determine the test cases that lead the exception handling routines to run [31]. In [33] it is claimed that aspect-oriented programming improves reliability by its nature providing direct control over exception handling.

Another way of improvement of quality and reliability of software systems is the code-inspection [34]. In the literature, there are examples of check-lists for improvement of quality of code-inspection process [32].

2.2.3 Project Management and Reliability

A direct use of software reliability studies appears in deciding the time when the product is ready to release [14, 36]. According to the current level of reliability, the amount of necessary testing is determined from the software reliability models by making use of failure data. By this way, it is also possible to measure cost of certain amount of increase in the reliability in terms of time, budget and man-hours [17].

Reference [3] presents a valuable discussion on how software development models affect the overall reliability of a software system. The models investigated in that study are Waterfall Model, Classic Development Model, Prototyping Approach, Spiral Model, Incremental Development Model, and Cleanroom Model. Among these, Waterfall Model is criticized for not allowing the solution of an inherent problem noticed in later phases, which increases the cost of reliability [16]. The problem of Classic Development Model with respect to reliability is stated to be the inefficiency of the model to help customer in determination of requirements in a clear manner. In that study, Prototyping Approach is suggested for improvement of quality and reliability since it provides feedback from the customer and actual users of the system. It is also indicated that Risk Analysis actions performed in each cycle of Spiral Model contributes to quality and reliability of the software system. According to [3] it is advisable to employ Incremental Development Model if specific functions/modules of the product have more strict reliability requirements.

2.2.4 Metric Collection Systems

If reliability is essential, then it has to be controllable. The necessary control process has to be based on observations or measurements. Because the raw material of these measurements may be defined differently from one organization to another, a generalized method of observation or measurement is needed. Metric collection systems are the answers to this need.

The process of creation of a software metric collection system is defined by [37] as of six successive steps. These steps are:

1. Documentation of the software development process
2. Statement of the purpose of the metric collection system

3. Determination of the metrics required to be collected in order to reach specific purposes
4. Identification of the data to be collected
5. Definition of the procedures to obtain data from the organization and projects
6. Coding of the designed overall system.

Ramakrishnan [38] reviews the general approaches to design of metric collection systems and remarks that two major approaches exist:

1. Fenton's Method: There exist three classes of software entities that may be measured; these are Process, Product, and Resources.
2. Hetzel's Bottom-Up IOR Model: Hetzel considers Input, Output, and Result as entities that may be measured.

Ramakrishnan's claim is that there is a duality between these two major approaches; Process, Product, and Resource of Fenton are equivalent to Hetzel's Input, Output, and Result, respectively. In that work basic principles of measurement systems are applied to a case where data is collected via manual forms, and entered to the system by a supervisor. Among these forms are Gannt charts, list of software products, log sheets of the members of the development team, and contracts. The findings are output in a report format. Once the findings are output as a report, they are considered as records; that is why the findings need to be validated before entered to the system. That gives birth to the problem that for large projects it is not easy for a supervisor to go through every single finding, and on the other hand, it is not feasible to automate the metric collection process completely without approval of the supervisor. For this reason, current state of the project can support non-objective criteria.

Another considerable metric collection system is reported by Chen [39]. Goal-Question-Metrics (GQM) is employed as guidance. The reason why GQM is selected for the guidance purposes is its suitability for goal-driven studies. The system has a client-server architecture, where client –side is regarded as "assistant agent", and server-side is "cooperative agent". Server actions are defined in terms of agent roles (like "Goal Identifier Agent", "Metrics Definer Agent", etc.). Client-side is mainly responsible for input-output interactions with the users of the system. All server-side actions (roles) operate collectively according to the inputs of the client-side and the goals to process relevant questions. That way, business goals are transformed into measurement goals, and a software measurement plan is defined in terms of objectives, description, implementation foresights and sustained operation of measurement.

Offen [40] points out a common mistake made by the companies that plan to employ metric collection systems: it is usually the case that the organizations first determine what metrics can be collected, and then define procedures to collect those. What would be the ideal case is to determine which metrics comply with the business goals, and then to define a complete process for measurement and metric collection. An algorithm proposed in that study to avoid that problem is as follows:

1. Understand business strategy
2. Name goals, risks, and sub-strategies
3. Determine factors that affect the success of the process
4. Define specific and neat development goals
5. Put forward questions
6. Define measures
7. Define the procedures to collect metrics from the development team
8. Review this process iteratively

Another software measurement system is reported by Eralp [41]. Metrics are selected according to measurement categories and specific organizational issues and goals, and are classified as Schedule Measures, Product Quality Measures, Resource and Cost Measures, and Size and Stability Measures. Next, responsibilities and priorities are assigned for each measure, and resources and collection mechanisms for these metrics are determined. The results are monitored via graphical user interfaces with different visual realizations.

In the next chapter, a metric collection system is formed in the light of information presented in this chapter. For this purpose, process improvement goals of the Company shall be discussed, metrics to be measured shall be determined, and existing process shall be revised to cover the measurements.

CHAPTER 3

PROPOSED MEASUREMENT SYSTEM

This chapter covers the development of a measurement system and modification of existing processes to include this measurement system. For this purpose, original processes of the Company are modeled via computer tools, as presented in APPENDIX A. Then, process improvement goals of the company are determined by conducting interviews with the stakeholders of the processes and the Company. Afterwards, the metrics to be collected to implement these goals are determined and examined with the stakeholders. After that, the details of the metric collection system are formed, and cost analysis is performed. Finally, modified processes are modeled and presented in APPENDIX B.

3.1 Process Improvement Goals of the Company

In the light of information presented in previous chapter, the first task of generation of a metric system was decided to be determination of process improvement goals of the company. Interviews conducted with the Chief of the Quality Assurance Division and a member of that division specialized in software projects have shown that principal goals of process improvement studies of the Company focus on so called 'Basic Processes' that define

- Organizational Management Activities (Strategic Planning, Management Review, Budget Planning, Correspondence, Business Development, Purchase),
- Project Management Activities (Project Planning, Project Monitoring),
- (Hardware) Production Activities,
- Infrastructure Maintenance Activities,
- Quality Assurance Activities,
- Quality Control Activities,
- Organizational Education Activities.

Due to the short history of the organization, the primary concern is to monitor the processes mentioned above, and those that define activities on (software) development are of secondary concern. The policy of the organization is to first stabilize Basic Processes, and then to improve development processes. For this reason there is no concrete aim in improving the development processes. This fact makes it harder to modify existing process definitions to collect a wider range of metrics on processes and the products.

Nevertheless, there exists a tendency to determine the level of reliability of software products. This tendency takes its roots from the customer expectations. The fact that current process definitions do not provide data on reliability of software products is considered as a weakness and a work-group named "Software Reliability Work Group" (SRWG) was formed for the purpose of determination of problematic issues in the software development procedures of the organization regarding software reliability and preparation of a proposal in order to draw a guideline for the removal or improvement of them. Models of process definitions for software development processes, namely Software Development Process, Requirements Elicitation

Process, Software Design Process, Coding Process, and Software Testing Process, are presented in APPENDIX A.

3.2 Determination and Classification of Metrics to be Collected

The starting point of work of Software Reliability Work Group, which is formed by representatives from Systems Engineering Division, Quality Assurance Division, Software Development Division and Modeling and Simulation Division, was the understanding of software reliability metrics commonly agreed in the software industry. The list and explanations of those metrics were obtained from IEEE Std 982.2-1988 [1].

Below, we evaluate the metrics presented in the document according to procedures that currently run at the Company. 24 out of 39 metrics are decided to be considerable for company goals. Software Reliability Work Group decided to scale selected metrics according to their 'availability' and 'relevance' to company procedures. The results are tabulated in Table 1. 'Availability' (A.) field in Table 1 indicates the availability of the metric with scaling explained as:

1. Currently being collected
2. Can be collected after minor modifications in procedures
3. May be collected only if there is a specific need in a project
4. Is not being collected and will not be collected

Table 1 - Evaluation of the Metrics for the Company

#	Metric Name	A.	R.	Ava*Rel
1	Fault-days number	1	2	2
2	Functional test coverage	1	1	1
3	Cause and effect graphing	4	4	16
4	Requirement traceability	1	1	1
5	Defect indices	4	3	12
6	Error distribution(s)	4	3	12
7	Software maturity index	1	2	2
8	Man hour per major defect detected.	3	3	9
9	Number of conflicting requirements	2	2	4
10	Software science measures	4	4	16
11	Run reliability	4	3	12
12	Design structure	3	3	9
13	Mean time to discover the next K fault	3	2	6
14	Software purity level	2	4	8
15	Requirement compliance	3	4	12
16	Test coverage	1	2	2
17	Residual fault count	4	3	12
18	Testing sufficiency	2	2	4
19	Failure rate	1	2	2
20	Software documentation & source listing	3	2	6
21	RELY(Required Software Reliability)	3	3	9
22	Software release readiness	3	3	9
23	Completeness	2	2	4
24	Test accuracy	2	3	6

Another field present in the table is ‘Relevance’ (R.), which is a measure of how tightly a metric is related to data requirements of work packages in the projects. The scale for relevance is as follows:

1. Must be collected
2. Potentially required in future projects
3. May be required upon customer request
4. Will never be required upon a direct request

In this study, we have decided to consider only those metrics with A*R value lower than or equal to 2. The metrics satisfying this condition are summarized in Table 2. The metric Fault-Days Number in Table 2 is defined in [1] as the number of days that passes before faults are removed from the software product. For better usage of this metric, average of this metric for all faults must be calculated.

Functional Test Coverage indicates what percent of functional requirements of the system under development has a corresponding test definition. It is used to determine if all the functional requirements are guaranteed to be tested.

Requirements Traceability is considered as a metric that helps determining what functional requirements are defined in the system under development per customer requirement. It may be regarded as the ratio of functional requirements to original requirements; thus deviation from unity must be avoided as neither missing nor additional requirements are desired.

Software Maturity Index is a measure of how stable a software product is. It can be measured by counting changes in a software product from one baseline to other. In cases where this is not possible, an alternative may be to count number of faults determined between two baselines. While from one point of view large number of faults determined indicates a more mature software product as faults are removed, it should be noticed that important percent of faults are determined in testing phase, and thus determination of faults in earlier phases may be a sign of existing faults that may be determined in testing as it is not known what percentage of existing faults are detectable.

Table 2 - Metrics to be Collected

#	Metric Name	A	R	A*R	Explanation
1	Fault-days number	1	2	2	Fully met via YUB Forms.
2	Functional test coverage	1	1	1	No direct correspondence. Technical Review Form and Test Result Form may be used for this purpose.
3	Requirements traceability	1	1	1	Technical Review Form and Test Result Form may be used for this purpose.
4	Software maturity index	1	2	2	Because software changes are not inspected in function level, Configuration Control Procedure needs revising.
5	Test coverage	1	2	2	Content of Technical Review Form may be adequate. It is determined by evaluating the test definitions with respect to requirements and the user.
6	Failure rate	1	2	2	Data can be collected by adding an extra field to YUB Forms.

In IEEE Std. 982.2-1988, Test Coverage metric (TC) is formulated as

$$\mathrm{TC} = \frac{\text{(number of defined features)}}{\text{(number of desired features)}} \times \frac{\text{(number of functions tested)}}{\text{(total number of functions)}} \times 100$$

According to this formulation, Test Coverage metric can be employed only after the design phase as both "number of defined features" and "total number of functions" are expected to vary until the end of design phase. As seen from the above equation, in order to calculate Test Coverage metric, number of defined features and number of desired features must precisely be known, which is possible only if the organization employs precautions to guarantee traceability from user requirements to product design. Also, calculation of Test Coverage metric requires the information of what percent of implemented functions are actually tested. This indicates the correlation with Test Coverage and Functional Test Coverage. For these reasons, it is possible to claim that in an organization that monitors Requirements Traceability and Functional Test Coverage, the infrastructure to monitor Test Coverage exists.

Failure Rate is a function of occurrences of failures within time. When a software product is tested, the occurrence times of failures are recorded and total number of failures is plotted as a function of time. For proper use of this metric, coding phase must be completed.

An interview has been conducted with Project Leaders and Software Quality Assurance Team Leader to determine if there exist a mechanism or a formulation to use the metrics mentioned above in decision making or report generation. The interview has showed that there is no such formulation as the Company currently lacks statistically meaningful set of data (metrics in this case) from previous or current projects. It is declared that available data is interpreted as a specific case of a specific project.

3.3 Formation of Metric Collection System

The process definitions altered to cover the metric collection system are presented in APPENDIX B. These models are for Software Development Process, Requirements Elicitation Process, Software Design Process, Coding Process, and Software Testing Process. For reasons that will be explained in Chapter 4, the modification is limited to changes defined by the requirements of the metrics Software Maturity Index, and Functional Test Coverage. As explained in Section 3.2, current processes readily meet the requirements of the metrics Fault-Days Number, and Requirements Traceability, there is no specific need for a change. Moreover, as discussed previously, monitoring of the metric Failure Rate does not assume any modification to the processes. This implies that the only metric that the requirements of which are not met is the Test Coverage metric.

Subsections of this section discuss modifications regarding each metric to be collected.

3.3.1 Modifications for Requirements of Software Maturity Index

As implied by the definition of Software Maturity Index, software products must be evaluated after baselines if software maturity is concerned. To improve the performance of the software development processes with respect to the metric

Software Maturity Index, a "Software Maturity Matrix" must be created after baselines defined by current process definitions are formed. Among these baselines that this work is concerned about are Software Functional Baseline, Software Design Baseline, and Software Product Baseline. Software Functional Baseline is formed at the end of Requirements Elicitation Phase, right after Software Requirements Specification is prepared. Software Design Baseline is formed after integration tests are defined at the end of Design Phase. Finally, Software Product Baseline is formed on delivery, after acceptance testing. Thus, in the modified processes, Software Maturity Matrix is formed and inspected after

1. Software Requirements Specification is prepared,
2. Requirements Elicitation is complete,
3. Design is complete.

To form Software Maturity Matrix, all the changes made on software products since previous baseline is formed shall be listed in a column. In the corresponding rows of the second column, the proof of need of the change and related explanations shall be given. Requirements affected by these changes shall be listed in the third column.

Each time the Software Maturity Matrix is created, a technical review must be performed in order to see if all the changes are indeed required changes, and all required changes are performed to modify all related functional requirements. If a fault is determined in the matrix, it should be reported via YUB, and be removed, and the Software Maturity Matrix must be updated. This process must be performed until all the faults appearing in the Software Maturity Matrix are removed.

Modifications regarding Software Maturity Index can be traced in Figure 18 (Modified Requirements Elicitation Process), Figure 19 (Modified Software Design Process), and Figure 21 (Modified Software Testing Process) in APPENDIX B, where the modifications are emphasized in bold frames.

3.3.2 Modifications for Requirements of Functional Test Coverage

In order to improve the performance of the software development processes with respect to the metric Functional Test Coverage, a "Functional Test Coverage Matrix" must be created right after the integration test are defined at the end of Design Phase. In the first column of the Functional Test Coverage Matrix shall be the functional requirements. The second column shall list associated modules of the software product. Finally, tests assigned to functional tests shall be listed in the third column.

This matrix is used to determine if existing modules meet the functional requirements and if a test method is associated with that module. That way it is guaranteed if the customer needs are transformed into design elements and each element –and thus customer need- is tested. If an improper entry is found in the matrix, the situation is reported via YUB, and related corrections are performed in the product after Design Review. An important fact that needs to be pointed out at this point is that while the essence of creation of this matrix is implicitly performed in the Company, they are not explicitly mentioned in procedures. Thus, addition of these stages mentioned above is completely realistic and does not affect the nature of company procedures.

Modifications regarding Functional Test Coverage can be traced in Figure 19 (Software Design Process) in APPENDIX B.

3.4 Cost of Modifications of the Procedures

The cost of modifying the process definitions and procedures to meet the requirements of Table 2 is analyzed in this section.

The costs expressed as man*hours are calculated in the light of information provided by Software Quality Assurance Team Leader, and the Chief of Systems Engineering Division.

3.4.1 Fault-Days Number

As explained in 3.2, Fault-Days Number indicates the number of days that faults spend in software products, and can be monitored in case of ongoing projects as well as completed ones. Such information is especially useful for future or ongoing projects if data on completed projects exist. The "life" of faults can be regarded as a performance index of software development process. If statistically adequate data is available, it would be possible for projects managers to suggest expectations of data of determination and removal of faults. That way it would be possible to outline a project calendar with narrower uncertainties, which in turn improves the effectiveness of project planning process.

It may be expected to avoid rework as much as 10 days * 2 men * 20% of a day * 8 hours a day = 32 man*hours per project. In other words, recording Fault-Days Number for each project would save up to 16 man*hours in future projects. On the other hand, according to Table 2 no modification to the current development process definitions is required; thus Fault-Days Number comes at no cost.

3.4.2 Functional Test Coverage

For properly monitoring Functional Test Coverage, Technical Review Form and Test Result Form must be revised. Once the organization is able to monitor Functional Test Coverage, ratio of faults determined to faults that exist in the software product will increase as possibility of delivery of a functional unit without testing is reduced. Moreover, excessive testing, or test duplication will be avoided, yielding reduced testing costs.

Assuming that currently 1 functional unit in a single project undergoes duplicated testing, 10 days * 2 men * 20% of a day * 8 hours / day = 32 man*hours per project is saved in case Functional Test Coverage is monitored.

The modification costs are limited to revision of two forms in total. Assuming that it takes one employee for a period of three workdays to revise a form, total cost is 2 forms * 1 man * 3 days / 1 form * 8 hours / day = 48 man*hours.

3.4.3 Requirements Traceability

Requirements traceability is one of the major concerns of the division of Systems Engineering at the Company. Moreover, in all of the software projects developed at the Company, the customer asks for an official evidence of requirements traceability. As software reliability can be judged in terms of conformance to performance requirements [45], the metric 'Requirements Traceability' serves as a measure of to what extent reliability analysis and testing is performed. Thanks to that fact, with no effort to change current procedures it will be possible to gain confidence with requirement traceability works, and to decrease rework effort that frequently occurs in case of a change in project documents and the product itself. In a typical project conducted at the Company, the rework effort can be calculated as 12 documents * 3 revisions * 1 man * 8 hours = 288 man * hours per project.

3.4.4 Software Maturity Index

This metric can be monitored after the Configuration Control Procedure is revised accordingly. For proper interpretation of the metric, either changes or faults in software products from one baseline to the next one are counted.

An expected outcome is parallelism between software maturity index and software reliability. It may be possible to gain confidence with reliability predictions or estimations, and to avoid further effort for reliability improvement work. This way, it would be possible to save up to 10 days * 2 men * 20% of a day * 8 hours / day = 32 man*hours.

The cost of being able monitor this metric is that of revision of Configuration Control Procedure, which may be approximated as 5 days * 1 man * 8 hours / day = 40 man*hours.

3.4.5 Test Coverage

Provided that the metrics Requirements Traceability and Functional Test Coverage can be monitored, Test Coverage can be monitored at no cost, as explained in Section 3.2.

The benefit of monitoring Test Coverage is that it indicates if all the desired features are implemented and tested completely. That way, customer satisfaction is guaranteed, and thus a more reliable product is delivered. Monitoring of Test Coverage does not provide saving for development costs.

3.4.6 Failure Rate

Failure Rate is the metric that enables quantitative analysis of product reliability. As explained in Chapter 2 in detail, the input of reliability growth models is the failure rate of a product. For this reason, in case software reliability of a software system has to be explicitly expressed in a quantitative manner, failure-time information should be recorded in system tests.

Current software development process readily assumes that this information is recorded. Hence, there is no need for a modification of the process definitions. Monitoring of Failure Rate does not provide saving for development costs.

3.4.7 Summary of Modification Costs

Table 3 presents a summary of information given in 3.4.1 to 3.4.6. It indicates that a total of 88 man * hours of work would result in a save of 384 man * hours per project. It should be noticed that while the total cost is to be spent once, the savings will be folded in each project completed.

Table 3 – Modification Costs Summary

#	Metric	Cost (man*hour)	Savings (man*hour)
1	Fault-Days Number	0	32 per project
2	Functional Test Coverage	48	32 per project
3	Requirements Traceability	0	288 per project
4	Software Maturity Index	40	32 per project
5	Test Coverage	0	N/A
6	Failure Rate	0	N/A
Total		88	384 per project

In the next chapter, evaluation of the modifications proposed in this chapter shall be discussed, and operation principles and structure of the computer tool developed for this purpose will be described in detail.

CHAPTER 4

PROPOSAL EVALUATION

In this chapter, first of all, alternatives for the evaluation of the system described in Chapter 3 shall be discussed. Then, the computer tool developed for the evaluation of the system and its features shall be described in detail. After the explanation of general flow of the tool, simulation philosophy and associated work shall be described.

4.1 Method for Evaluation

4.1.1 Evaluation Alternatives

It is widely accepted that the best way of evaluation of process improvement proposals and modifications is application of modified procedures in a real project or organization [41]. By actually applying the improved processes, it is possible to observe direct effects of the proposed system. As the time progresses and projects with different characteristics are developed with the new system, problematic points are detected and removed; real statistical data about the system and projects is obtained and used to further improve the system.

During the development of the proposed measurement system it was planned to perform mentioned modifications in the Company procedures and gather statistical data from the projects being developed by the Company. Due to re-organization of the Company, however, computer simulation became the primary alternative for the evaluation of the proposed system as it permits construction of the system, without affecting the organizational structure, and requires limited resources. Consequently, computer simulation of the system reduces the cost and managerial risk of modifications.

It is possible with the simulation tool to characterize a set of programmers working for the Company, define a project by providing its size and estimated development duration, and to perform simulations as if the Company develops the defined projects, with defined programmers. The response of the simulation tool to different cases can be analyzed by defining and simulating different projects with different sizes and different development duration estimates, and by altering workloads of the programmers. More importantly, the simulation tool can be customized to account for variations in the ability of the Company to detect faults existing within a software product.

The tool generates output files that contain information about the fault content of the software product developed, development duration, and size of the final product. That way, user is provided with the ability to compare reliability levels of the products developed with the current software development process of the Company and with the one proposed in Chapter 3.

4.1.2 Limitations of the Simulation

While during the development of the system proposed in Chapter 3, the entire set of needs of the Company is considered and all the requirements are met, the simulation introduces certain limitations. In Chapter 3, six metrics were suggested to be monitored. These are:

1. Fault-Days Number
2. Functional Test Coverage
3. Requirements Traceability
4. Software Maturity Index
5. Test Coverage
6. Failure Rate

Among these metrics Functional Test Coverage, Requirements Traceability, and Test Coverage necessitates detailed statistical data on the structure of software products in function level such as desired functionalities, defined functionalities, number of modules, total number of functions in a module, number of module functions tested. To be able to include this data in a simulation, statistically meaningful number of sample projects must be available; otherwise the data would be misleading. Unfortunately, there is only limited number of software projects developed by the Company, and thus limited data exists. For this reason these metrics are not feasible for simulation purposes. Thus, Functional Test Coverage, Requirements Traceability, and Test Coverage will not be monitored in the simulation. Nevertheless, absence of this data does not affect the dependability of the simulation tool as the tool treats all the metrics independently, and allows addition of new metrics when adequate data is provided by the Company.

In addition, Failure Rate is a metric that may only have significance in the testing phase; it does not affect the design phase. This is why it will not be monitored in the simulation. As it is aimed in this study to examine how reliability of the products is affected by the proposed modifications, exemption of Failure Rate, which does not have effect on the design, is acceptable.

4.2 Simulation Tool Outline and Features

For the purpose of simulation of the proposed measurement system described in Chapter 3, a computer tool with a user-friendly interface is developed. With this computer tool, the user may define different projects for which software development processes of the Company will be simulated; that is, the program is supposed to simulate the Company processes as if a defined project is to be developed via these processes. To achieve this, user may define programmers that will participate in the development process. Moreover, the user may modify the information regarding the projects and programmers.

Once a project is defined and opened for simulation, the computer tool runs the computer models of both original processes of the Company and those proposed in Chapter 3. For this purpose, process models in APPENDIX A and APPENDIX B are of special importance as they constitute the basis of simulation flow. The simulation results are stored in different text files for inspection.

The simulation of development of a project as well as creation of project and programmer information is achieved via user commands. The user commands are summarized in Table 4. Level-1 commands shown in Table 4 either perform an atomic task or introduce a list of atomic tasks. Level-2 commands are all atomic tasks.

The relations of the database files and user commands are summarized in Figure 2. Below, the main commands are overviewed.

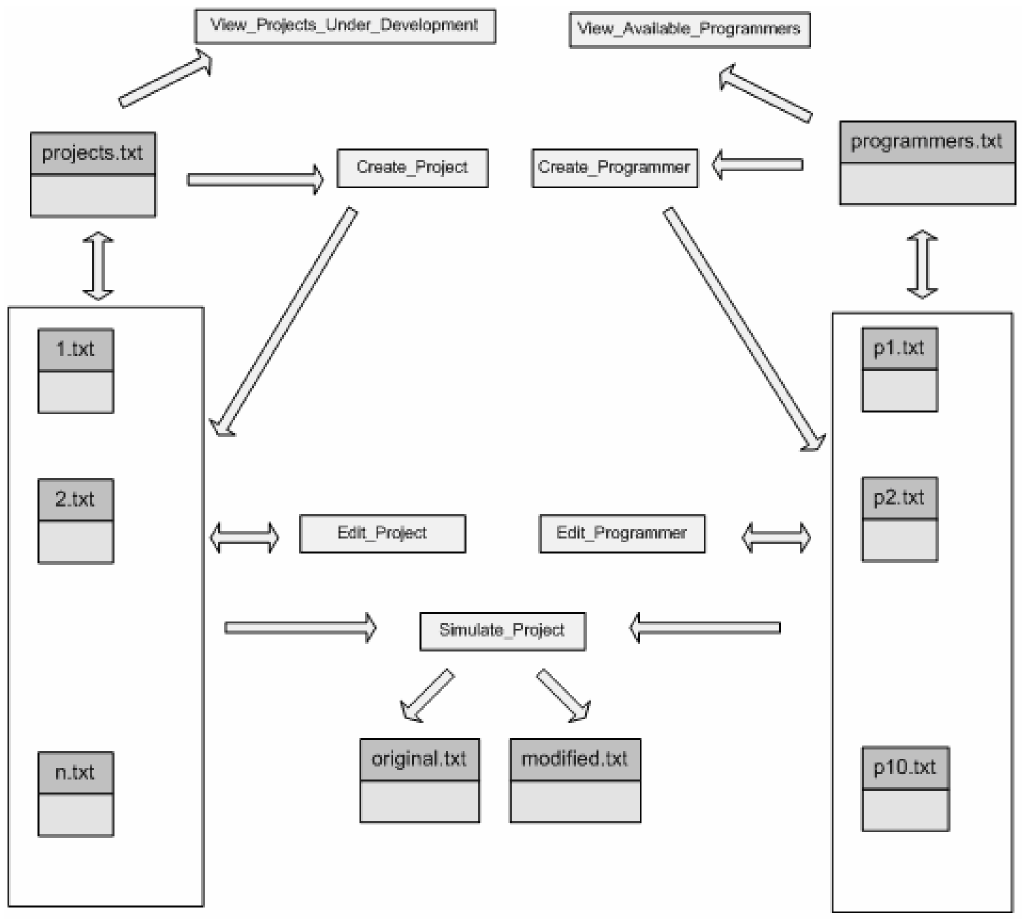


Figure 2 - Relations of Database Files and User Commands

Table 4 – User Commands

User Commands	
Level-1 Commands	**Level-2 Commands**
Create Project	
Edit Programmers Pool	Create Programmer
	View Available Programmers
	Edit Programmer
Open Project	View Projects Under Development
	Edit Project
	Simulate Project
Quit Program	

4.2.1 "Create Project" Command

If the user selects "Create Project" command, the program prompts the user to provide "Name of the Project", "Project Manager", and "Estimated Size of the Project" as thousands lines of code. For the user to choose the project manager among available programmers, the command "View Available Programmers" is automatically processed.

The project information obtained from the user is stored in a text file that bears a name which is equivalent to the order of the project; the information of the first project created is stored in the file 1.txt, and that of the nth project is stored in the file [n].txt. The order information is obtained from the file projects.txt that stores the order and the name of the projects created. The flowchart of "Create Project" command is given in Figure 3.

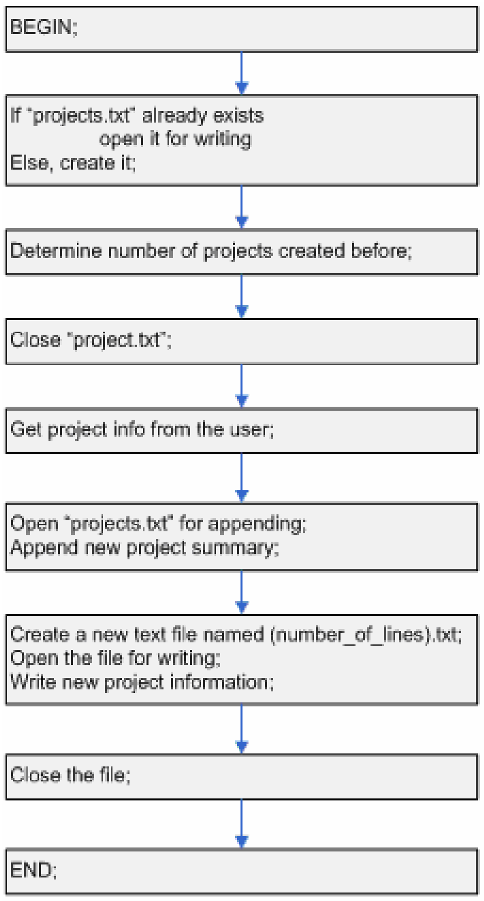

BEGIN;
If "projects.txt" already exists open it for writing Else, create it;
Determine number of projects created before;
Close "project.txt";
Get project info from the user;
Open "projects.txt" for appending; Append new project summary;
Create a new text file named (number_of_lines).txt; Open the file for writing; Write new project information;
Close the file;
END;

Figure 3 - Flowchart of "Create Project" Command

4.2.2 “Create Programmer” Command

When the user selects “Create Programmer” command, the program expects the user to enter a “Name”, a “Surname”, and the “Experience” of the programmer in years. The programmer information obtained from the user is stored in a text file that bears a name which is equivalent to the order of the programmer preceded by the letter “p”. The information of the first programmer created is stored in the file p1.txt, and that of n^{th} programmer is stored in the file p[n].txt. The order information is obtained from the file programmers.txt that stores the order and the name of the programmers created. The flowchart of “Create Programmer” command is given in Figure 4.

The simulation program permits creation of up to ten programmer profiles. This amount is realistic when the software development team of the Company is considered.

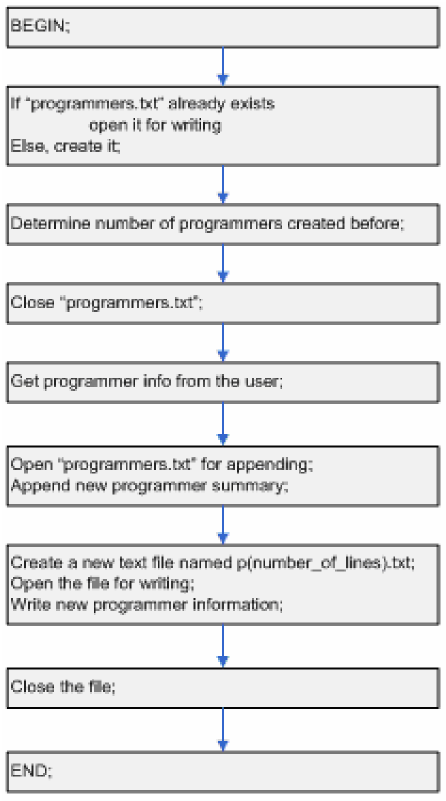

BEGIN;
If "programmers.txt" already exists
open it for writing
Else, create it;
Determine number of programmers created before;
Close "programmers.txt";
Get programmer info from the user;
Open "programmers.txt" for appending;
Append new programmer summary;
Create a new text file named p(number_of_lines).txt;
Open the file for writing;
Write new programmer information;
Close the file;
END;

Figure 4 - Flowchart of "Create Programmer" Command

4.2.3 "View Available Programmers" Command

When the user selects "View Available Programmers" command, contents of the files with names starting with the letter "p" is listed according to the format in Table 5. The flowchart of "View Available Programmers" command is given in Figure 5.

Table 5 – Output of "View Available Programmers"

No	Name of the Programmer	Surname of the Programmer	Experience
1	Programmer_1_Name	Programmer_1_Surname	Exp_1
.	.	.	.
.	.	.	.
.	.	.	.
n	Programmer_n_Name	Programmer_n_Surname	Exp_n

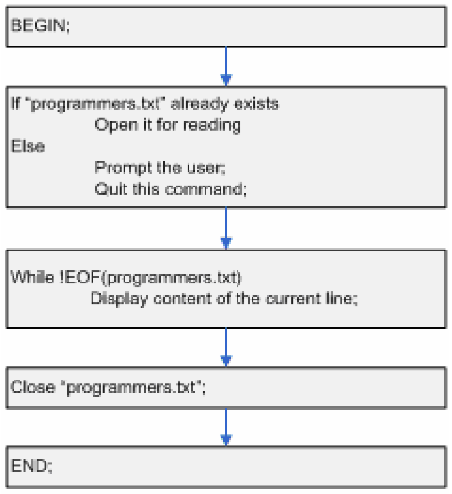

BEGIN;
If "programmers.txt" already exists
Open it for reading
Else
Prompt the user;
Quit this command;
While !EOF(programmers.txt)
Display content of the current line;
Close "programmers.txt";
END;

Figure 5 - Flowchart of "View Available Programmers" Command

4.2.4 "Edit Programmer" Command

In case the user desires to modify information of a specific programmer, the command "Edit Programmer" should be employed. When called, this command calls "View Available Programmer" command automatically to force the user to choose the programmer whose information is to be modified. Once the programmer is selected, the program displays current information and prompts the user to enter new information. When the new information is obtained from the user, it replaces the content of the file that keeps obsolete information. The flowchart of “Edit Programmer” Command is given in Figure 6.

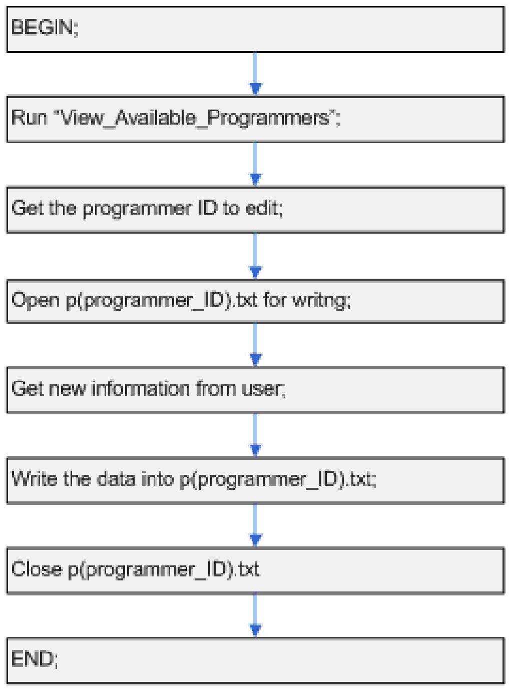

BEGIN;
Run "View_Available_Programmers";
Get the programmer ID to edit;
Open p(programmer_ID).txt for writng;
Get new information from user;
Write the data into p(programmer_ID).txt;
Close p(programmer_ID).txt
END;

Figure 6 - Flowchart of “Edit Programmer” Command

4.2.5 “View Projects Under Development” Command

The operation of “View Projects Under Development” command is similar to that of “View Available Programmers” command. Once this command is selected, contents of the files that store project information are listed according to the format in Table 6. The flowchart of “View Projects Under Development” Command is given in Figure 7.

Table 6 - Output of “View Projects Under Development”

No	**Project Name**	**Project Manager Name**	**Project Manager Surname**	**Estimated KLOC**	**Estimated Duration**
1	P1_Name	P1_Mng_Name	P1_Mng_Surn	P1_KLOC	P1_Duration
.	.	.	.	.	.
.	.	.	.	.	.
.	.	.	.	.	.
n	Pn_Name	Pn_Mng_Name	Pn_Mng_Surn	Pn_KLOC	Pn_Duration

4.2.6 “Edit Project” Command

“Edit Project” command acts in a similar fashion as “Edit Programmer” command. When called by the user, it calls “View Projects Under Development” so as to force the user to choose the project that is to be updated. When a project is selected, current information about the project is displayed and new information is accepted.

The new information obtained from the user replaces obsolete content of the project file. The flowchart of “Edit Project” Command is given in Figure 8.

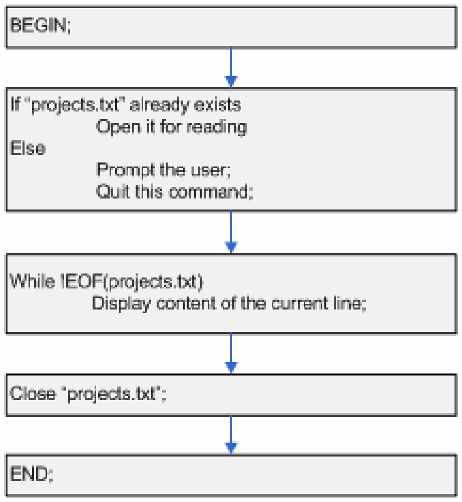


Figure 7 - Flowchart of "View Projects Under Development" Command

4.2.7 "Simulate Project" Command

"Simulate Project" command constitutes the heart of the tool. The projects and programmer profiles created and updated with other commands are resources of this command.

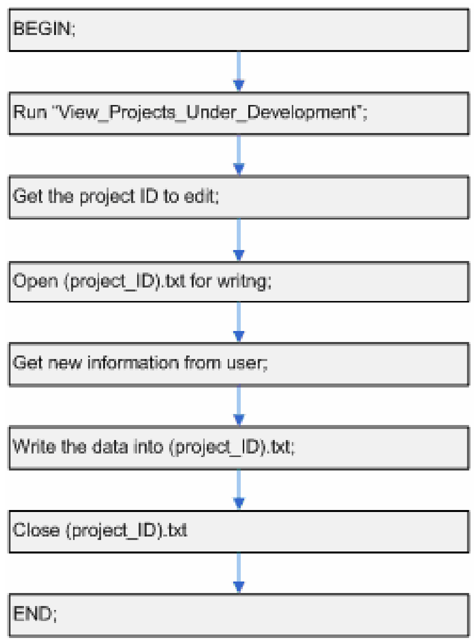

BEGIN;
Run "View_Projects_Under_Development";
Get the project ID to edit;
Open (project_ID).txt for writng;
Get new information from user;
Write the data into (project_ID).txt;
Close (project_ID).txt
END;

Figure 8 - Flowchart of "Edit Project" Command

When a project is opened and "Simulate Project" command is selected, for a realistic operation of the simulation, the tool expects two more inputs from the user: Fault Finding Ability, and Company Workload Index. Fault Finding Ability is an index that may have values in the range [0..100] and points what percent of the faults in software products could be determined when software development process is executed. This index should be obtained from the user as there is no available statistical data due to the fact that post-developmental procedures of the Company (e.g. Maintenance Procedure) are not defined within the organization yet. Company Workload Index may have values in the range [0..10] and indicates how busy the Company is. The larger the Company Workload Index, the larger the possibility that programmers are reserved for other business activities of three days long is. To account for real life situations, this probability is assigned a Rayleigh distribution with mean 3.

“Simulate Project” command performs two different simulations on the same resources: the first one employing original software development procedures of the Company, the second one with proposed measurement system in Chapter 3.

When simulating with original procedures, the tool monitors Total Time Elapsed (TTE), Number of Faults Introduced within Total Time Elapsed (NoFI), Number of Faults Determined within Total Time Elapsed (NoFD), and estimate of total lines of code after each step of development procedure (KLOCE). In the case of simulation of modified procedures, that is the software development process with measurement system of Chapter 3, in addition to TTE, NoFI, NoFD, and KLOCE, the tool also monitors identities of faults determined in each step of development process and their removal time from the software products. The flowchart of “Simulate Project” Command is given in Figure 9.

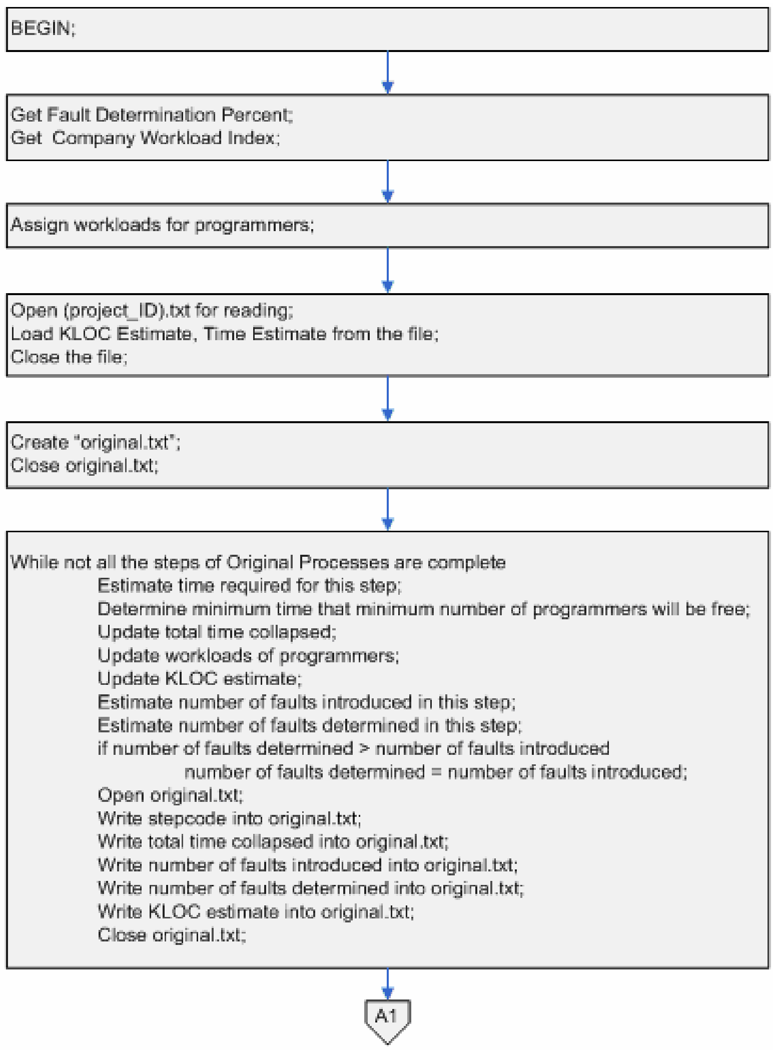
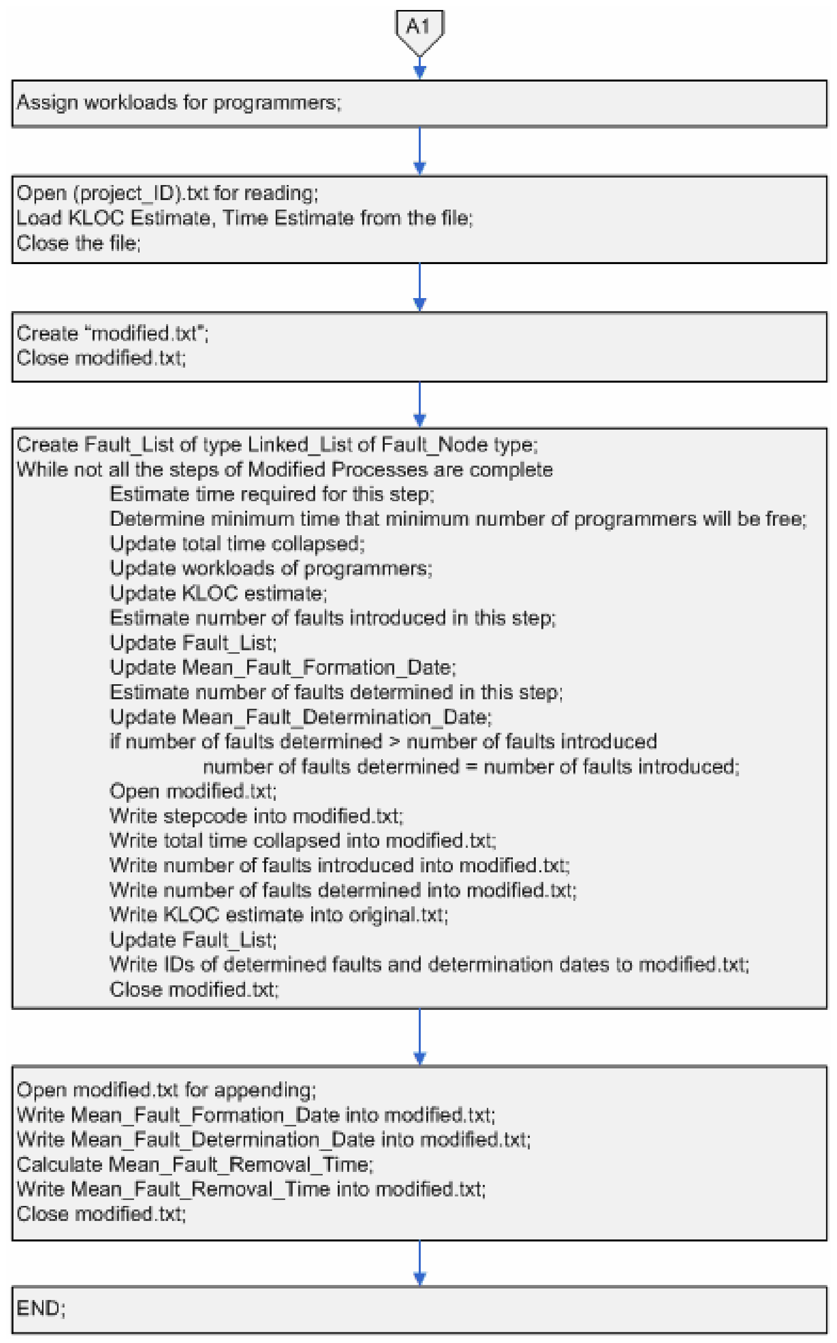

BEGIN;
Get Fault Determination Percent;
Get Company Workload Index;
Assign workloads for programmers;
Open (project_ID).txt for reading;
Load KLOC Estimate, Time Estimate from the file;
Close the file;
Create "original.txt";
Close original.txt;
While not all the steps of Original Processes are complete
Estimate time required for this step;
Determine minimum time that minimum number of programmers will be free;
Update total time collapsed;
Update workloads of programmers;
Update KLOC estimate;
Estimate number of faults introduced in this step;
Estimate number of faults determined in this step;
if number of faults determined > number of faults introduced
number of faults determined = number of faults introduced;
Open original.txt;
Write stepcode into original.txt;
Write total time collapsed into original.txt;
Write number of faults introduced into original.txt;
Write number of faults determined into original.txt;
Write KLOC estimate into original.txt;
Close original.txt;
A1

Figure 9 - Flowchart of "Simulate Project" Command



```
Assign workloads for programmers;

Open (project_ID).txt for reading;
Load KLOC Estimate, Time Estimate from the file;
Close the file;

Create "modified.txt";
Close modified.txt;

Create Fault_List of type Linked_List of Fault_Node type;
While not all the steps of Modified Processes are complete
        Estimate time required for this step;
        Determine minimum time that minimum number of programmers will be free;
        Update total time collapsed;
        Update workloads of programmers;
        Update KLOC estimate;
        Estimate number of faults introduced in this step;
        Update Fault_List;
        Update Mean_Fault_Formation_Date;
        Estimate number of faults determined in this step;
        Update Mean_Fault_Determination_Date;
        if number of faults determined > number of faults introduced
                number of faults determined = number of faults introduced;
        Open modified.txt;
        Write stepcode into modified.txt;
        Write total time collapsed into modified.txt;
        Write number of faults introduced into modified.txt;
        Write number of faults determined into modified.txt;
        Write KLOC estimate into original.txt;
        Update Fault_List;
        Write IDs of determined faults and determination dates to modified.txt;
        Close modified.txt;

Open modified.txt for appending;
Write Mean_Fault_Formation_Date into modified.txt;
Write Mean_Fault_Determination_Date into modified.txt;
Calculate Mean_Fault_Removal_Time;
Write Mean_Fault_Removal_Time into modified.txt;
Close modified.txt;

END;
```

Figure 9 - Flowchart of "Simulate Project" Command (Continued)

4.3 Simulation Details

The simulation approach applied in this work assumes that each step of software development procedures presented in APPENDIX A and APPENDIX B affects organizational "resources", and "fault content" of software products. Organizational resources are time and programmers, whereas fault content contains number of faults present in software products, number of faults determined within a step, time spent on removal of a specific fault, and the ratio of faults determined (ROFD), which is defined to be the ratio of number of faults determined to number of faults introduced.

In other words, whenever a project is opened for simulation, corresponding steps of original processes and proposed system are simulated with the assumption that each step causes the following actions:

1. Time is consumed
2. One or more programmers are assigned a specific task, thus will not be available for a certain time
3. Size of the project being developed changes
4. Number of faults present in the software product being developed changes
5. Number of determined faults changes

Thus, each time a step is to be simulated, the simulation tool is internally fed with mean values of

1. Required time for the execution of the step
2. Number of programmers required for the execution of the step

3. Percent change in size
4. Number of faults created in the step
5. Number of faults determined in the step.

This way, the simulation gains the ability to reflect such realistic effects as overshoot of size and development duration estimations, which were declared by Company officials to be typically around 35% for size, and 25% for time. Subsections of this section describes the details of how above five actions are simulated in the tool.

4.3.1 Consumption of Time and Programmers

The mean time required by a step is determined by taking average of corresponding time information in projects conducted by the Company. In order to guarantee randomness as in realistic projects, Rayleigh distribution was chosen as the probability distribution function as it avoids negative ranges.

The time requirement obtained this way -say T1- constitutes the minimum time that is required to complete the given step. In reality, however, a step requires certain number of employees available to perform a given task and related statistical data is fed to the simulation tool for this purpose. For a task requiring n employees, employee profiles existing in the database are scanned to determine the minimum time -T2- before n employees will be ready to perform a new task. Therefore the simulation points that T1+T2 much time has to pass to complete a given step, instead of T1.

Once time requirement is determined, employee profiles are updated according to the new time information: If busy time of an employee was shorter than time elapsed in a step, the employee is assigned a new task according to 4.2. If busy time of an employee was longer than time elapsed in a step the busy time is shortened by T1+T2.

4.3.2 Variation of Project Size

After the employee profiles are updated, the simulation tool checks if the step currently being processed introduces a change in the estimate of total lines of codes of the project. If it does, percent change in the total lines of codes, which is accepted as an input from real projects of the Company, is used to stochastically modify lines of code estimate. In this modification, Gaussian distribution [44] is assumed as a change may mean a decrease in total lines of codes and an increase as well.

4.3.3 Formation and Detection of Faults

The next task performed in the simulation after project size is updated is the determination of number of faults determined in this step. The nominal value of number of faults determined in a step for real projects of the Company can readily be obtained from YUBs. Since number of faults introduced in a step cannot precisely be known, the method discussed in Section 4.2.7 is used to determine that number.

As mentioned in Section 4.2.7, faults introduced, determined, and removed in modified processes are required to bear IDs. For this purpose, two classes are designed: Fault_Node and Linked_List.

4.3.3.1 Fault_Node Class

This class is designed to represent “faults” in the simulation. A summary of this class is given in Table 7. When a fault is determined, the day it is determined shall be set with the function Set_Day_Determined. The method Set_Day_Removed randomly assigns KLOC for the fault removed, and employs COCOMO method to calculate the removal time.

4.3.3.2 Linked_List Class

This class is designed to prepare a list of “faults” of type Fault_Node in the simulation. A summary of this class is tabulated in Table 8.

Table 7 - Summary of Fault_Node Class

Private Data			
Name	**Type**		
ID	integer		
Day_Created	integer		
Day_Determined	integer		
Day_Removed	integer		
Public Data			
Name	**Type**		
Next_Entry	Fault_Node *		
Methods			
Name	**Type**	**Input Name**	**Input Type**
Insert_After_This	void	Insert_This	Fault_Node *
Delete_After_This	void	void	
Return_ID	integer	void	
Return_Day_Created	integer	void	
Return_Day_Determined	integer	void	
Return_Day_Removed	integer	void	
Set_Day_Determined	void	Day_Determined	integer
Set_Day_Removed	void	void	

4.4 Other Issues Regarding Simulation

There are issues that must be discussed as they have direct effects on the way simulation is performed. These are:

- Size and duration estimation
- Random number generation

Table 8 - Summary of Linked_List Class

Public Data			
Name	**Type**		
Root_Node	Fault_Node *		
Last_Node	Fault_Node *		
Private Data			
Name	**Type**		
None			
Methods			
Name	**Type**	**Input Name**	**Input Type**
Delete_Node_With_ID	void	ID	integer
Add_Node	void	ID	integer
		Date_Created	integer
Search_With_ID	Fault_Node *	ID	integer

4.4.1 Size and Duration Estimation

Statistical information on projects conducted by the Company indicates that an average project is conducted by 30 person*month in 12 months (252 workdays) yielding 8554 LOC.

For simulation purposes, this data is used as a guideline to determine required resources of a project to be simulated.

If Basic COCOMO model is assumed [43], following parameters may be used for estimation of resource requirements:

$$\text{Effort} = a_b * \text{KLOC}^{bb} \text{ (person*month)}$$
$$\text{Duration} = c_b * \text{Effort}^{db} \text{ (month)}$$

where

$$a_b = 2.28,\ b_b = 1.2,\ c_b = 2.55,\ d_b = 0.445$$

Throughout the simulation, above parameters are used whenever size and development duration estimations are performed.

4.4.2 Random Number Generation

Assignment of values to random variables constitutes an important part of this work as it directly affects the performance of the simulation tool in terms of its realism. For the purpose of generation of random variables, first of all, two uniformly distributed random variables were created each time an assignment is to be done. These random variables are then transformed into polar coordinates to obtain Gaussian distributed random variables [44]. In cases when Rayleigh distributed random variables were required, which is the case when time resources are consumed, a uniformly distributed random variable was generated and then transformed with the following equation:

$$rv = \sqrt{\left|\log(\text{uni\_rv} \bullet \text{uni\_rv})\right|}$$

where rv is the resultant random variable and uni_rv is the uniformly distributed random variable. rv is multiplied by the mean to obtain desired values.

Histograms of outputs of functions for generation Gaussian and Rayleigh distributed random variables with mean 1 are given in Figure 10 and Figure 11 respectively. Both functions were run 65536 times.

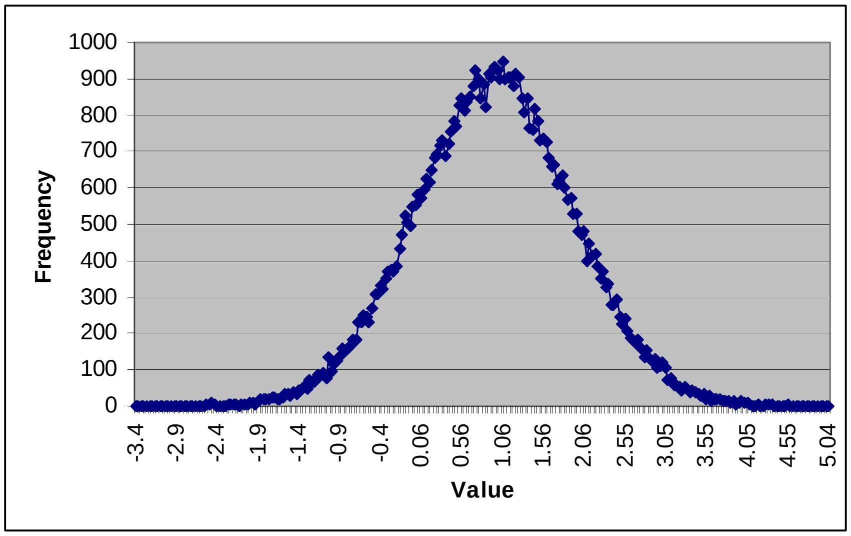


Figure 10 - Histogram for Gaussian Random Variable Generation

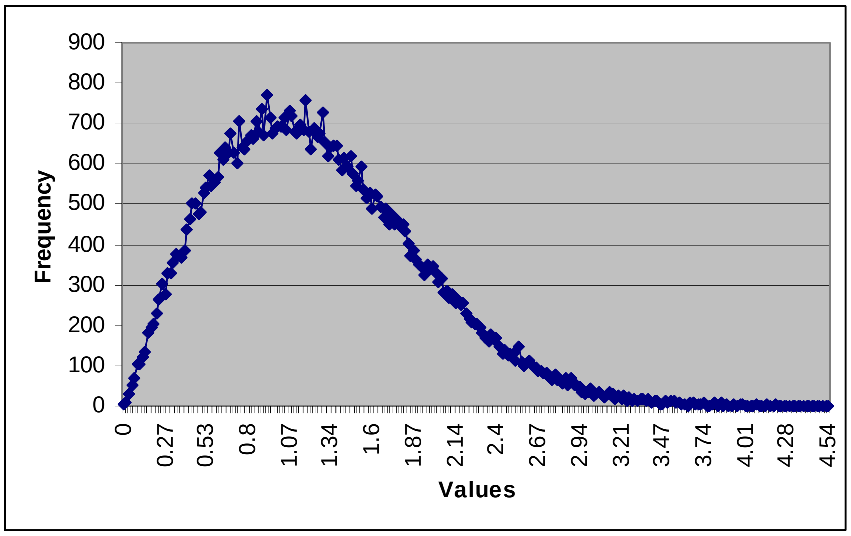


Figure 11 - Histogram for Rayleigh Random Variable Generation

4.5 Output Data and Output Format

The final step of the simulation of a step is the recording of simulation results into a text file with the format given in Table 9. The output of simulation of original processes of the Company is stored in a file named "Original.txt", whereas that of proposed system is stored in "Modified.txt".

"Step" column shown in Table 9 contains an integer code that indicates what step of software development process was simulated. The lists of step codes for original and modified procedures are presented in APPENDIX C, and APPENDIX D, respectively. In the output files, "TTE" column contains workdays that have passed until the end of a step. "NoFI" column indicates number of faults introduced until the end of a step. In the "NoFD" column is number of faults determined until the end of a step. “KLOC” column is the one where estimate of total thousand lines of codes is given. Finally, “Faults Determined” column contains two kinds of information: identity of the fault determined in a step and removal time of that fault indicated in parentheses.

Table 9- Format for Simulation Results

Step	**TTE**	**NoFI**	**NoFD**	**KLOC Estimate**	**Faults Determined**
.					
.					
n	TTEn	NoFIn	NoFDn	KLOCEn	IDi(RTi),IDj(RTj),...IDk(RTk)
.					
.					

The removal time is calculated using the same COCOMO parameters used for the simulation with the assumption that a fault necessitates a developmental activity taking certain amount of lines of code. That number of lines of code is assumed to be a random variable changing uniformly between 0 and 600, where limits are arithmetic mean of statistical data from the Company. It should, however, be pointed out that there is an offset of 2 weeks, that is 10 workdays due to other organizational procedures employed in the Company. Thus, the time information obtained with COCOMO parameters should be added to 10 to yield removal time.
Another assumption in the calculation of removal time is that determination of a fault is an interrupt to the software development process, and no matter how busy the employees are, priority is given to the removal of faults determined.

When simulation of all the steps is completed, if software development procedure with proposed measurement system is being simulated, additional items that will be used in interpretation of results are appended to the output. These items are:

1. Arithmetic mean of date of formation of each fault (MFD)
2. Arithmetic mean of date of determination of each fault (MDD)
3. Arithmetic mean of duration of removal of each fault (MRT)

The difference between MDD and MFD yields mean time that a fault spends in the software product without being determined (MTbFD). Sum of MTbFD and MRT yields mean time that a fault spends in the software product from its formation to its removal (MTbFR). The purpose of monitoring MTbFD and MTbFR is to show that it is possible to calculate cost of each software fault to end up with an estimate of cost of correction of faults in software products. It should be noted that MTbFR is equivalent to the metric Fault-Days Number. While in a more advanced simulation tool MRT and and MTbFR would be important outputs, in this study, they are estimated according to the COCOMO technique based on uniformly distributed re-work sizes per fault. As such, they have not been included in the simulation outputs.

In the next chapter, the results obtained by operating the simulation tool are presented and discussed.

CHAPTER 5

RESULTS AND DISCUSSION

In this chapter, first the performance criteria for the simulation tool and test conditions shall be defined. Then, the output shall be discussed to determine if proposed measurement system and the simulation tool yields consistent results.

5.1 Generation of Simulation Results

In order to properly evaluate the performance of the simulation tool, first of all it should be decided what portion of the output data actually have significance in the evaluation of the measurement system regarding reliability. Then, the input characteristics must be defined to extensively test the system. In addition, the number of times that these experiments are going to be repeated should precisely be determined. These issues are discussed in the subsections of this section.

5.1.1 Data of Interest

As mentioned in the previous chapter, when the simulation tool is run two text files are generated: Original.txt, and Modified.txt. The file Original.txt stores the simulation results of original software development procedures of the Company, whereas the content of the file Modified.txt is the results of procedures with proposed measurement system.

Statistically valuable part of these files is the rows corresponding to the last step of development procedures; that is, total time elapsed from the beginning of the project, total number of faults introduced, total number of faults determined, final KLOC value. Also, for the modified procedures, MDD, MFD, and MRT are of importance.

5.1.2 Project Characteristics

Another important point in generation of simulation results is the determination of characteristics of projects for which software development procedures are executed. To be able to analyze the response of the simulation tool to each project characteristic each attribute of a sample project is assigned three different values within their meaningful ranges. The summary of projects simulated is presented in Table 10.

The projects P1, P2, and P3 are simulated to determine the response of the simulation tool to changes in estimated duration of projects. Response of the tool to KLOC estimate changes is investigated via the projects P1, P8, and P9. The projects P1, P4, and P5 are simulated to monitor response of the tool to Company Workload Index variations. Finally, the projects P1, P6, and P7 are used for the purpose of examining the response of the tool for different Fault Finding Ability values.

Table 10 - Simulated Projects

Project #	Estimated KLOC	Estimated Duration (Workdays)	Company Workload Index	Fault Finding Ability
P1	9	250	0	60
P2	9	375	0	60
P3	9	500	0	60
P4	9	250	5	60
P5	9	250	10	60
P6	9	250	0	75
P7	9	250	0	90
P8	12	250	0	60
P9	18	250	0	60

5.1.3 Determination of Number of Runs

Because of the fact that calculations performed within the simulation have stochastic components, it is not possible to interpret the results with one run. Instead, the simulation must be repeated for "enough times". Uncertainty of "enough times" may be removed thanks to the method described in [42], where a method for determination of number of trials to end up with a mean that lies within a known neighborhood of the actual mean with a given confidence level is described. In this work, the neighborhood is limited to ±5% and the confidence level is set to 95%, meaning that the probability that the arithmetic mean of simulation results lies within ±5% of actual mean is 95%.

The parameters that have been considered in the application of this method are

1. TTE of original procedures
2. NoFI of original procedures
3. NoFD of original procedures
4. KLOC Estimate of original procedures
5. TTE of modified procedures
6. NoFI of modified procedures
7. NoFD of modified procedures
8. KLOC Estimate of modified procedures
9. MFD of modified procedures
10. MDD of modified procedures
11. MRT of modified procedures
12. MTbFD of modified procedures
13. MTbFR of modified procedures

The simulation is repeated until all thirteen variables satisfy the confidence rule described in the preceding sub-section.

5.2 Verification of Simulation

Dependability of the simulation tool can be judged by comparing its output for a given project to actual developmental data available for this project. For this purpose it is wise to run the simulation tool for the sample project mentioned in Section 4.4.1, which requires 30 person*months, 252 workdays of development duration, and consisting of 8554 LOC (rounded to 9 KLOC).

The summary of the results of the simulation can be found in Table 11, where the project is designated as P1. The results show that simulation yields a development duration of around 313 days, and size of 12.5 KLOC. The ratio of simulated development duration to estimated duration is 24.2%, and that of simulated size to estimated size is 38.9%. It should be noticed that these overshoots are close to the ones mentioned in Section 4.3, namely, 35% for size, and 25% for time estimations. Hence, it is safe to assume that the simulation tool does simulate the software development process of the Company.

5.3 Discussion of Results

Simulations are performed in the light of explanations given in 5.1. The summary of simulations of nine projects is presented in Table 11. For the purpose of illustration of how Table 11 was formed from the simulation results, last ten rows of simulation summary of Project 1 defined in Table 10 are presented in APPENDIX E. As visualized in APPENDIX E, Run Number, TTE, NoFI, NoFD, and KLOC for simulations of both original and modified processes are listed. In addition to these, MFD, MDD, MRT, MTbFD, and MTbFR results are also provided for simulation of modified processes. Later, the method mentioned in 5.1.3 is employed to calculate required number of runs to have all the variables within desired neighborhood with desired possibility. These run numbers are designated with n'. The simulation is performed repeatedly until all n' values are larger than or equal to the run number. When this condition is met, mean values of simulation variables are transferred to Table 11.

When Table 10 and Table 11 are evaluated together, it is observed that the simulation program acts consistently. To deepen the inspection of the response of the tool, groups of projects must be considered instead of single projects.

5.3.1 P1, P2, and P3: Response to Changing Duration

As mentioned previously, sample projects considered in the calculation of COCOMO model parameters had development duration of one year. For this reason it is currently not possible to analyze the effects of development duration on RODF in spite of the fact that RODF is expected to be improved with longer development duration.

Keeping KLOC estimate constant, an increase in estimated duration of the project must not affect the size of the project. Simulated duration must be the only variable to change. This, indeed, is the case as seen in the simulation results. According to Table 11, change in estimated duration only affects simulated duration for both original and modified processes.

Table 11 - Summary of Simulation Results of Nine Characteristic Projects

Project #		Original Process								Modified Process					
	Duration	NoFI	NoFD	KLOC	Percent NoFD/NoFI	Duration	NoFI	NoFD	KLOC	Percent NoFD/NoFI	MFD	MDD	MRT	MTbFD	MTbFR
P1	312.91	183.70	122.97	12.54	66.94	315.59	194.23	141.08	13.66	72.64	116.07	175.99	10.53	54.65	64.25
P2	458.23	181.11	119.73	12.27	66.11	456.95	189.02	138.19	13.32	73.11	169.1	257.02	10.65	59.9	67.15
P3	609.89	183.17	124.04	12.52	67.72	600.08	188.93	136.56	13.23	72.28	221.23	335.53	10.63	113.04	123.56
P4	370.27	182.75	123.55	12.55	67.61	370.88	192.57	139.23	13.62	72.30	134.58	206.16	10.64	60.56	69.56
P5	403.23	184.73	125.11	12.56	67.73	402.98	194.29	141.69	13.59	72.93	144.46	224.38	10.71	79.92	90.64
P6	312.04	147.02	122.70	12.40	83.46	314.87	154.30	139.76	13.65	90.58	115.55	175.98	10.63	60.43	71.05
P7	311.38	122.12	119.31	12.38	97.70	313.10	128.65	128.05	13.51	99.53	114.47	191.03	10.6	76.56	87.16
P8	353.61	239.01	155.50	15.78	65.06	352.96	250.69	176.72	16.88	70.49	128.74	197.76	10.68	54.62	63.07
P9	416.12	342.07	215.43	21.22	62.98	408.78	354.90	242.32	22.59	68.28	147.6	227.49	10.77	71.99	81.69

5.3.2 P1, P4, and P5: Response to Changing Workload Index

Effect of change of workload index is the limited increase of development duration in original and modified procedures. KLOC estimation and Ratio of Determined Faults, which was defined in Section 4.3, remain unchanged, as expected.

5.3.3 P1, P6, and P7: Response to Changing Fault Finding Ability

Inspection of simulation results for P1, P6, and P7 indicates that a change in Fault Finding Ability input, which was defined in Section 4.2.7, directly affects Ratio of Determined Faults, without affecting other simulation parameters. This is a consistent result as neither duration nor KLOC estimations are related with Fault Finding Ability. It is worth pointing that for a constant KLOC, number of determined faults remains constant while that of introduced tends to decrease regularly with increasing Fault Finding Ability.

5.3.4 P1, P8, and P9: Response to Changing KLOC Estimate

KLOC estimate may be considered to be the main parameter of the simulation as it constitutes the basics of COCOMO calculations. For this reason, even slight changes in KLOC estimates give rise to significant changes in other simulation parameters. This fact can be observed in the simulation results as well. When KLOC estimate is increased from 9 to 12, and then to 15, duration, NoFI, and NoFD increases accordingly. The increase, however, is not a linear relationship. When the KLOC estimate is doubled, duration is increased by a factor of 1.33. The reason for this is that, growth of duration follows COCOMO model where ratio of durations of n-KLOC and 2n-KLOC projects converges to 1.448, instead of 2. Variation of KLOC estimate within simulation steps is the reason that simulated ratio is 1.33, and not 1.448.

5.3.5 Overall Response Evaluation

For the purpose of evaluation of response of the simulation tool in terms of software reliability, it is wise to compare the resultant Percent NoFD / NoFI values of original processes to those of modified processes. Table 12 presents these values and the ratios of Percent NoFD/NoFI values of Original Processes to those of Modified Processes.

Inspection of table indicates that for all nine characteristic projects, the ratios of Percent NoFD/NoFI values of Original Processes to those of Modified Processes are less than unity. This fact, in turn, indicates that original processes are less promising in the sense that they are able to determined smaller ratio of existing faults in a software product. As the definition of software reliability and failure imply that faults present in a software product is a sign of lack of reliability, it is wise to claim that modified processes yield more reliable software.

Table 12– Percent NoFD/NoFI Values

Project #	Original Processes Percent NoFD/NoFI	Modified Processes Percent NoFD/NoFI	Ratio
1	66.94	72.64	0.92
2	66.11	73.11	0.90
3	67.72	72.28	0.94
4	67.61	72.30	0.94
5	67.73	72.93	0.93
6	83.46	90.58	0.92
7	97.70	99.53	0.98
8	65.06	70.49	0.92
9	62.98	68.28	0.92
AVERAGE	71.70	76.90	0.93

Moreover, the durations of characteristics projects when developed via original processes and modified processes are summarized in Table 13. It is seed from the table that a project is developed in almost the same durations. This means that the increase in software reliability is obtained almost at no cost. In addition to these, the productivity of the organization is increased slightly as number of LOC produced in a given time is increased by a factor of 7.9%, which is a fact indicated in Table 14.

In the next chapter, conclusions regarding this work and possible future studies shall be discussed.

Table 13 - Durations of Projects

Project #	Original Processes Duration	Modified Processes Duration	Ratio
1	312.91	315.59	0.99
2	458.23	456.95	1.00
3	609.89	600.08	1.02
4	370.27	370.88	1.00
5	403.23	402.98	1.00
6	312.04	314.87	0.99
7	311.38	313.10	0.99
8	353.61	352.96	1.00
9	416.12	408.78	1.02
AVERAGE	394.19	392.91	1.00

Table 14 - Sizes of Projects

Project #	Original Processes Project Size	Modified Processes Project Size	Ratio
1	12.54	13.66	0.92
2	12.27	13.32	0.92
3	12.52	13.23	0.95
4	12.55	13.62	0.92
5	12.56	13.59	0.92
6	12.40	13.65	0.91
7	12.38	13.51	0.92
8	15.78	16.88	0.93
9	21.22	22.59	0.94
AVERAGE	13.80	14.89	0.93

CHAPTER 6

CONCLUSIONS

In this work three major tasks are accomplished. First, software reliability literature is examined; basic definitions of software reliability studies are understood; classification and relations of software reliability growth models are inspected; application areas of software reliability concepts are investigated.

As the second major task, software development procedures of a company are examined with software reliability being the primary concern; problematic issues are determined; expectations of the organization in terms of software reliability are determined; different metrics are evaluated according to their usefulness and availability, a selection is extracted accordingly, and in the light of information obtained, a new set of procedures with a measurement system is proposed.

In the final stage of this work, a computer program with a user-friendly interface is developed to simulate both original and modified procedures, to evaluate the benefits to be achieved via the proposed measurement-based control system.

6.1 Discussion of Findings

This study shows that reliability of software products can be increased without dramatically altering software development procedures, and at a low cost.

It is observed that the ability of the organization to determine faults existing in a software product is improved. This is achieved primarily by modifying existing processes to monitor the metrics Software Maturity Index, and Functional Test Coverage. This point is remarkable as this study does not aim to propose an improvement to the software development processes in general sense; rather it introduces minor modifications to existing processes with software reliability being the primary concern. This fact can be extracted from the simulation results if it is realized that while the development duration and hence costs remain constant, the reliability in increased.

6.2 Future Work

Simulation of proposed system enables evaluation of the system without actually modifying organizational structure and procedures, which may give birth to short chaotic periods and infeasible costs. With simulation, it is made possible to analyze the effects of modifications without managerial and economical risks. For organizations considering minor modifications in development procedures, it may be advisable to perform simulations of modifications before actual changes take place to find out defective points with the new set of changes.

It should, nevertheless, be noticed that the decision of performing a simulation instead of accomplishment of actual process improvement tasks introduces certain limitations.

The first limitation comes from the fact that the nature of simulations is limited to internally seeded set of inputs as it requires significant resources to design and implement a simulation tool fully characterized to real-world situations. For this reason, simulation may only provide an outline of what would happen in a realistic situation. The level of realism of the simulation tool may be increased if variety and depth of statistical data obtained from real projects are increased. This may also make it possible to implement simulation tools that monitor a broader range of functionalities. These functionalities may range from additional metrics to monitor to managerial activities as decision making and project management.

The second limitation is that simulations are performed on user defined projects, which are nothing but variations of a sample project obtained by linearly varying certain characteristics of the sample project. It would improve the dependability of this work if it were possible to compare the simulation to projects with diverse characteristics.

There exist alternatives to overcome the above limitations. One of these alternatives is the actual application of the proposed system to the Company, and the second one is the improvement of the simulation tool to cover all realistic effects and a more detailed definition of the organization, employees, and the projects.

Apart from simulation approach, a detailed project improvement project may be a significant choice for the improvement of reliability of software products developed in an organization. For this purpose, it would be wise to start with classification and analysis of faults determined so as to find out the roots of the problematic points with the development process and the development team. Once these points are determined, more specific solutions may be proposed and supported with related trainings.

REFERENCES

1. "IEEE Std 982.2-1988, IEEE Guide for the Use of IEEE Standard Dictionary of Measures to Produce Reliable Software", 1998

2. L. Rosenberg, T. Hammer, J. Shaw, "Software Metrics and Reliability", http://satc.gsfc.nasa.gov/support/ISSRE_NOV98/software_metrics_and_reliability.html
   Last Date Accessed: 25.08.2005

3. "MIL-HDBK-338B, Electronic Reliability Design Handbook", US DoD

4. J. Musa, A. Iannino, K. Okumoto, "Software Reliability: Measurement, Prediction, Application", 1990, McGraw-Hill

5. C. Huang, M. R. Lyu., S. Kuo, "A Unified Scheme of Some Non-Homogenous Poisson Process Models for Software Reliability Estimation", March 2003, IEEE Transactions on Software Engineering, Volume: 29 Issue: 3, pp. 261-269

6. S. Ramani, S. Gokhale, K. S. Trivedi, "SREPT: Software Reliability Estimation and Prediction Tool",
   http://www.ee.duke.edu/~ssg/papers/tools98.ps
   Last Date Accessed: 25.08.2005

7. M. R. Lyu, J. Schönwälder, “Web-CASRE: A Web-Based Tool for Software Reliability Modeling“,
http://www.cse.cuhk.edu.hk/~lyu/paper_ps/web-casre.ps
Last Date Accessed: 25.08.2005

8. S. Brocklehurst, B. Littlewood, “New Ways to Get Accurate Reliability Measures”, July 1992, IEEE Software, Volume: 9 Issue: 4, pp. 34-42

9. F. Lanubile, “Why Software Reliability Predictions Fail”, July 1996, IEEE Software, Volume: 13 Issue: 4, pp. 131-132, 137

10. B. Littlewood, “The Problems of Assessing Software Reliability”, 2000, Proceedings of the Safety Critical Systems Symposium

11. R. C. Tausworthe, M. R. Lyu, “A Generalized Technique for Simulating Software Reliability”, March 1996, IEEE Software, Volume: 13 Issue: 2, pp. 77-88

12. H. Pham, “Software Reliability and Testing”, 1995, IEEE Computer Society Press

13. Y. K. Malaiya; M. N. Li; J. M. Bieman; R. Karcich, “Software Reliability Growth with Test Coverage“, December 2002, IEEE Transactions on Reliability, Volume: 51 Issue: 4, pp. 420-426

14. M. C. K. Yang; A. Chao, “Reliability-Estimation and Stopping-Rules for Software Testing, Based on Repeated Appearances of Bugs”, June 1995, IEEE Transactions on Reliability, Volume: 44 Issue: 2, pp. 315-321

15. P. K. Kapur, R. B. Garg, “A Software Reliability Growth Model for an Error-Removal Phenomenon“, July 1992, Software Engineering Journal, Volume: 7 Issue: 4, pp. 291-294

16. P. J. Boland, H. Singh, “A Birth-Process Approach to Moranda's Geometric Software-Reliability Model”, June 2003, IEEE Transactions on Reliability, Volume: 52 Issue: 2, pp. 168-174

17. W. Ehrlich, B. Prasanna, J. Stampfel, J. Wu, “Determining the Cost of a Stop-Test Decision“, March 1993, IEEE Software, Volume: 10 Issue: 2, pp. 33-42

18. M. Şahinoğlu, “Compound-Poisson Software Reliability Model”, July 1992, IEEE Transactions on Software Engineering, Volume: 18 Issue: 7, pp. 624-630

19. K. Goševa-Popstojanova, K. S. Trivedi, “Failure Correlation in Software Reliability Models”, March 2000, IEEE Transactions on Reliability, Volume: 49 Issue: 1, pp. 37-48

20. C. Wohlin, U. Korner, “Software Faults: Spreading, Detection and Costs”, January 1990, Software Engineering Journal, Volume: 5 Issue: 1, pp. 33-42

21. S. L. Pfleeger, “Software Engineering: Theory and Practice”, 1998, Prentice Hall

22. M. Chen, M. R. Lyu, W. E. Wong, “Effect of Code Coverage on Software Reliability Measurement”, June 2001, IEEE Transactions on Reliability, Volume: 50 Issue: 2, pp. 165-170

23. J. D. Musa, “A Theory of Software Reliability and Its Applications”, September 1975, IEEE Transactions on Software Engineering, Volume: SE-1, No: 3, pp.312-327

24. B. Littlewood, “Software Reliability”, 1987, Blackwell Scientific Publications

25. W. W. Everett, “Software Component Reliability Analysis”, 1995, “Software Reliability and Testing”, Los Alamitos, California, IEEE Computer Society Press, pp.45-46

26. A.L. Goel, “Software Reliability Models: Assumptions, Limitations, and Applicability”, December 1975, IEEE Transactions on Software Engineering, Volume: SE-11 No: 12, pp. 1411-1423

27. M. R. Lyu, A. Nikora, “Applying Reliability Models More Effectively”, July 1992, IEEE Software, pp. 43-52

28. A. L. Goel, K. Okumoto, “Time-Dependent Error-Detection Rate Model for Software Reliability and Other Performance Measures”, August 1979, IEEE Transactions on Reliability, Vol. R-28 No. 3, pp. 206-211

29. J. Voas, “Assuring Software Quality Assurance”, May-June 2003, IEEE Software, Volume: 20 Issue: 3, pp. 48-49

30. R. L. Glass, “Defining Quality Intuitively”, May-June 1998, IEEE Software, pp. 103-107

31. J. Viega, J. Voas, “Can Aspect-Oriented Programming Lead to More Reliable Software?”, November-December 2000, IEEE Software, pp. 19-21

32. J. R. de Almeida Jr., J. B. Camargo Jr., B. A. Basseto, S. M. Paz, “Best Practices in Code Inspection for Safety-Critical Software”, May-June 2003, IEEE Software, Volume: 20 Issue: 3, pp. 56-63

33. R. Laddad, “Aspect-Oriented Programming Will Improve Quality”, November-December 2003, IEEE Software, pp. 90, 92

34. J. Barnard, A. Price, “Managing Code Inspection Information”, March 1994, IEEE Software, Volume: 11 Issue: 2, pp. 59 -69

35. V. T. Rokosz, “Long-Term Testing in a Short-Term World”, May-June 2003, IEEE Software, pp. 64-67

36. M. R. Lyu, S. Rangarajan, A. P. A. van Moorsel, “Optimal Allocation of Test Resources for Software Reliability Growth Modeling in Software Development”, June 2002, IEEE Transactions on Reliability, Volume: 51 Issue: 2, pp. 183 -192

37. “Creating a Metrics Program”, Software Productivity Center Inc., http://spc.ca/resources/metrics/
Last Date Accessed: 25.08.2005

38. S. Ramakrishnan, T. Menzies, M. Hasslinger, P. Bok, H. Mccharty, B. Devakadadcham, D. Moulder, “On Building an Effective Measurement System for OO Software Process, Product and Resource Tracking”, November 1996, TOOLS Pacific, pp. 239 - 247

39. T. Chen, B. Homayoun Far, Y. Wang, “Development of an Intelligent Agent-Based GQM Software Measurement System”, August 2003, The First International Conference on Agent-Based Technologies and Systems, pp. 187-197

40. R. Offen, R. Jeffery, “Establishing Software Measurement Programs”, March-April 1997, IEEE Software, pp. 45-53

41. Ö. Eralp, “Design and Implementation of a Software Development Process Measurement System”, 2004, Master of Science Thesis, Middle East Technical University

42. Prof. Dr. S. Bilgen, 2005, Lecture Notes

43. B. W. Boehm, “Software Engineering Economics”, 1981, Prentice Hall

44. A. Papoulis, “Probability, Random Variables, and Stochastic Processes”, 1991, McGraw Hill

45. P. B. Crosby, “Quality is Free: the Art of Making Quality Certain”, 1979, McGraw Hill

APPENDIX A

THE CURRENT SOFTWARE DEVELOPMENT PROCESS

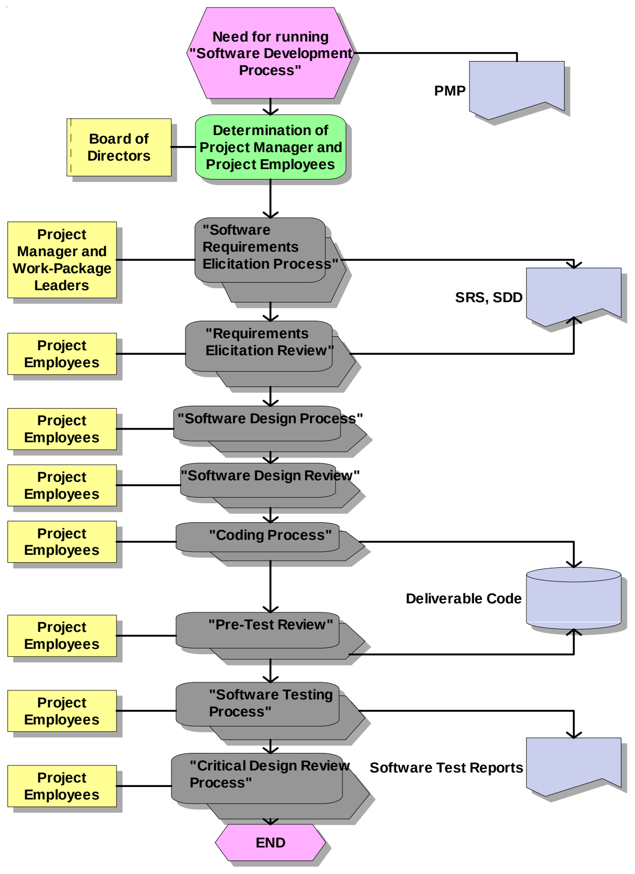

Need for running "Software Development Process"
PMP
Board of Directors
Determination of Project Manager and Project Employees
Project Manager and Work-Package Leaders
"Software Requirements Elicitation Process"
SRS, SDD
Project Employees
"Requirements Elicitation Review"
Project Employees
"Software Design Process"
Project Employees
"Software Design Review"
Project Employees
"Coding Process"
Deliverable Code
Project Employees
"Pre-Test Review"
Project Employees
"Software Testing Process"
Software Test Reports
Project Employees
"Critical Design Review Process"
END

Figure 12 – The Current Software Development Process

Requirements Elicitation Process

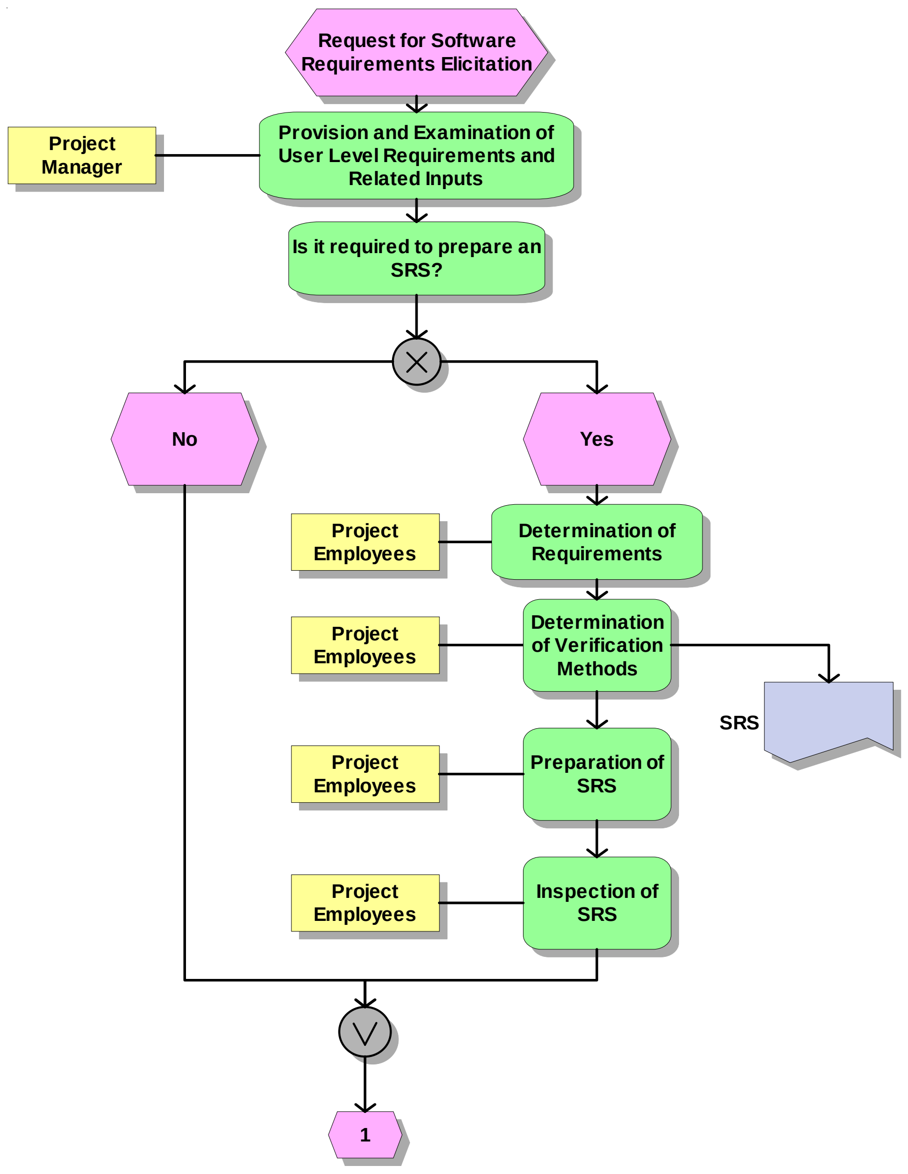

Request for Software Requirements Elicitation
Project Manager
Provision and Examination of User Level Requirements and Related Inputs
Is it required to prepare an SRS?
No
Yes
Project Employees
Determination of Requirements
Project Employees
Determination of Verification Methods
SRS
Project Employees
Preparation of SRS
Project Employees
Inspection of SRS
1

Figure 13 - Requirements Elicitation Process

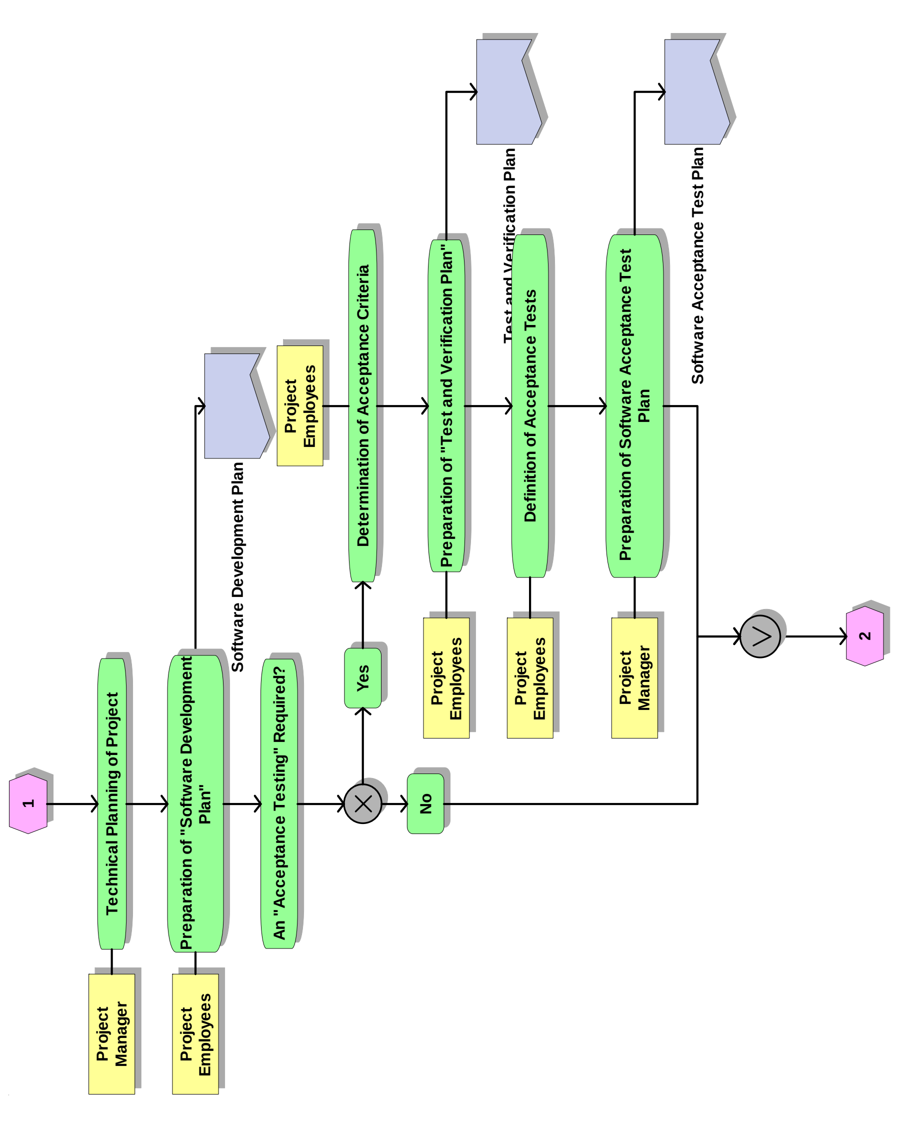

1
Technical Planning of Project
Project Manager
Preparation of "Software Development Plan"
Project Employees
Software Development Plan
An "Acceptance Testing" Required?
Yes
No
Determination of Acceptance Criteria
Project Employees
Preparation of "Test and Verification Plan"
Project Employees
Test and Verification Plan
Definition of Acceptance Tests
Project Employees
Preparation of Software Acceptance Test Plan
Project Manager
Software Acceptance Test Plan
2

Figure 13 - Requirements Elicitation Process (Continued)

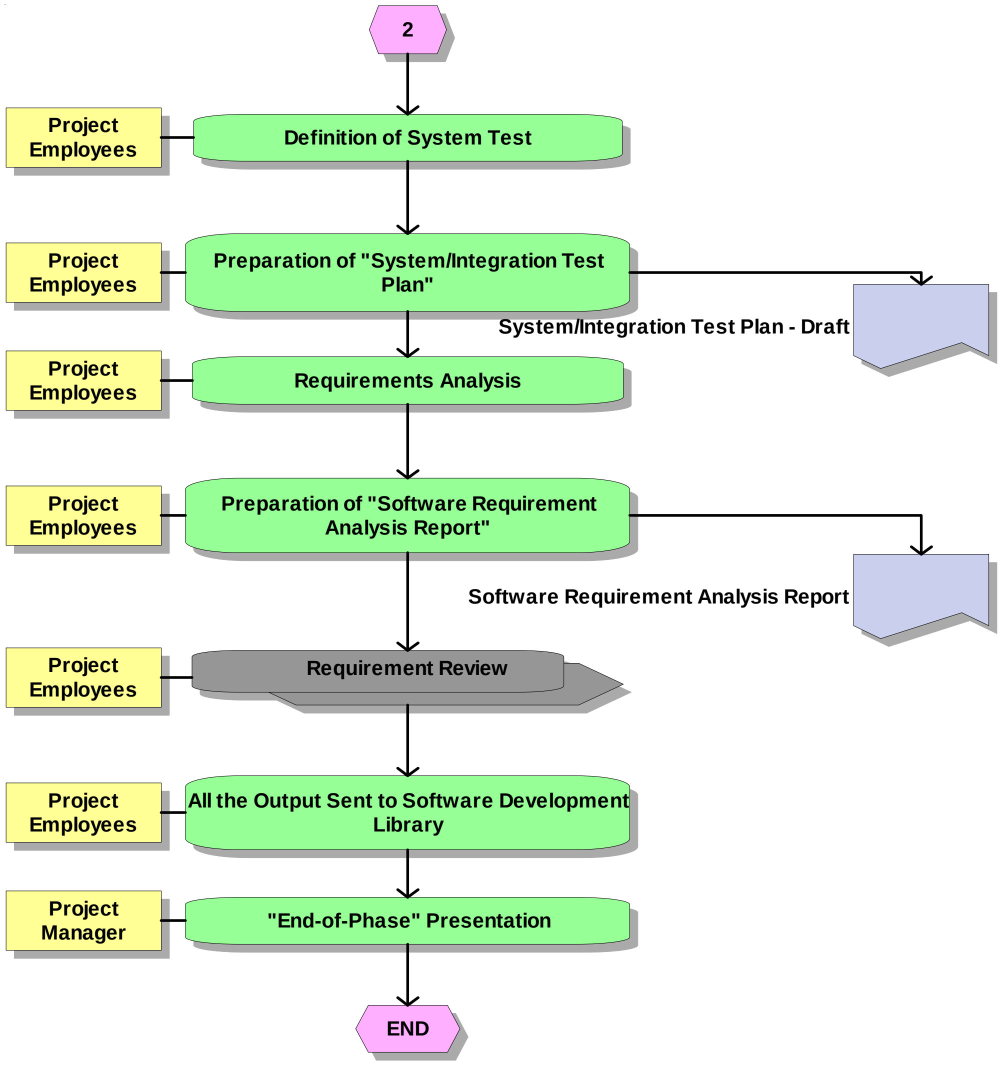

2
Project Employees
Definition of System Test
Project Employees
Preparation of "System/Integration Test Plan"
System/Integration Test Plan - Draft
Project Employees
Requirements Analysis
Project Employees
Preparation of "Software Requirement Analysis Report"
Software Requirement Analysis Report
Project Employees
Requirement Review
Project Employees
All the Output Sent to Software Development Library
Project Manager
"End-of-Phase" Presentation
END

Figure 13 - Requirements Elicitation Process (Continued)

Software Design Process

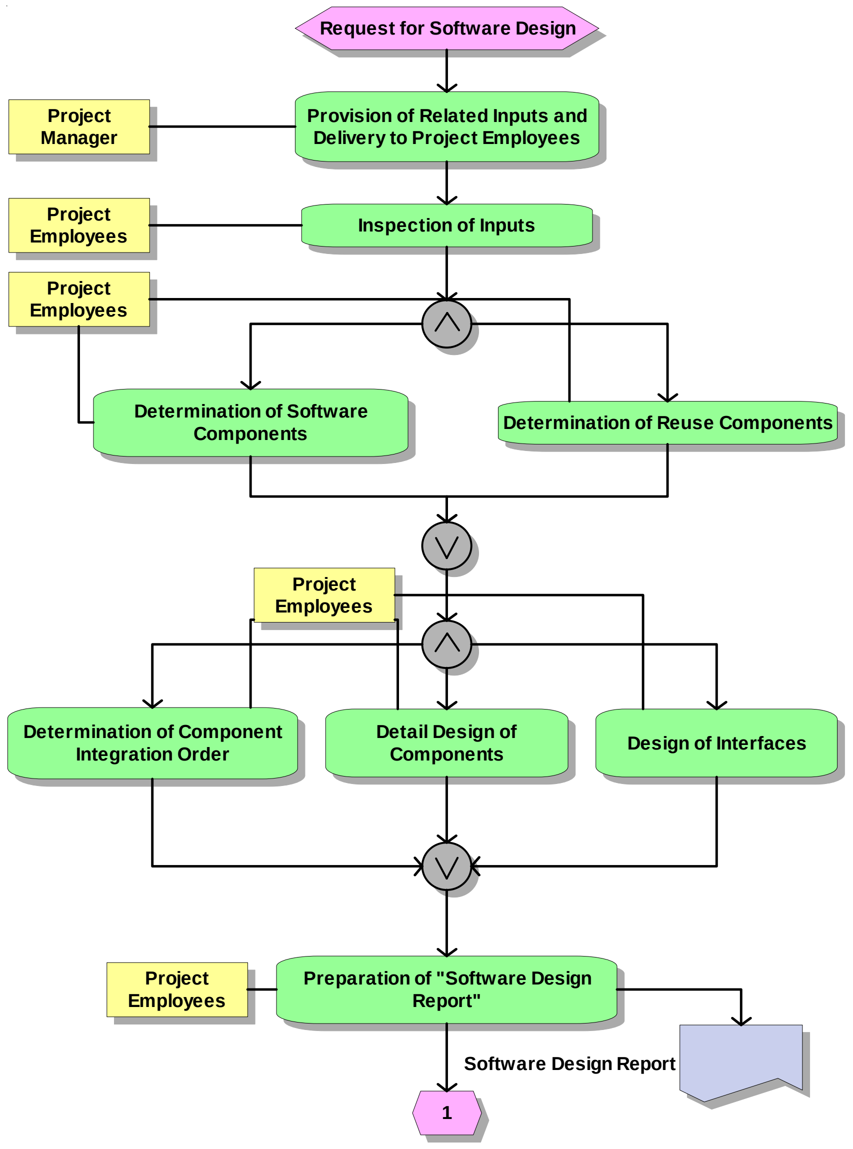

Request for Software Design
Project Manager
Provision of Related Inputs and Delivery to Project Employees
Project Employees
Inspection of Inputs
Project Employees
Determination of Software Components
Determination of Reuse Components
Project Employees
Determination of Component Integration Order
Detail Design of Components
Design of Interfaces
Project Employees
Preparation of "Software Design Report"
Software Design Report
1

Figure 14 - Software Design Process

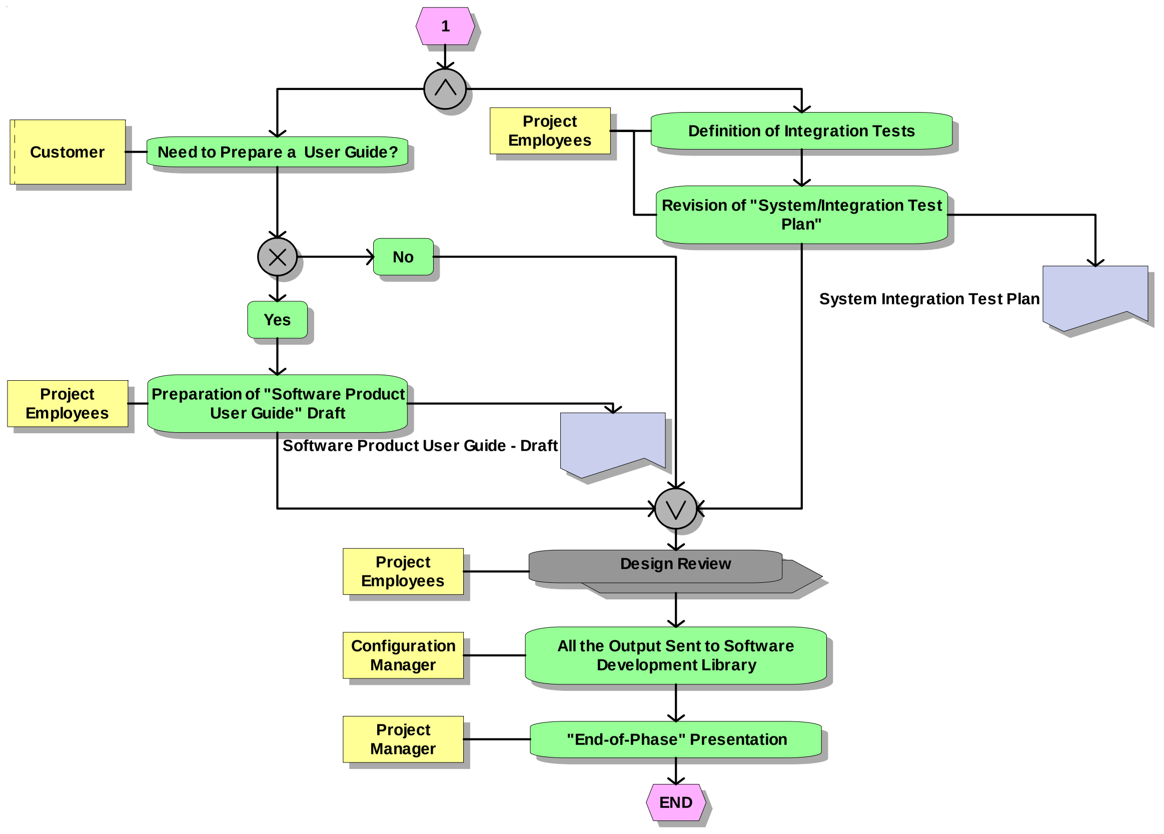

1
Customer
Need to Prepare a User Guide?
Project Employees
Definition of Integration Tests
Revision of "System/Integration Test Plan"
System Integration Test Plan
No
Yes
Project Employees
Preparation of "Software Product User Guide" Draft
Software Product User Guide - Draft
Project Employees
Design Review
Configuration Manager
All the Output Sent to Software Development Library
Project Manager
"End-of-Phase" Presentation
END

Figure 14 - Software Design Process (Continued)

Coding Process

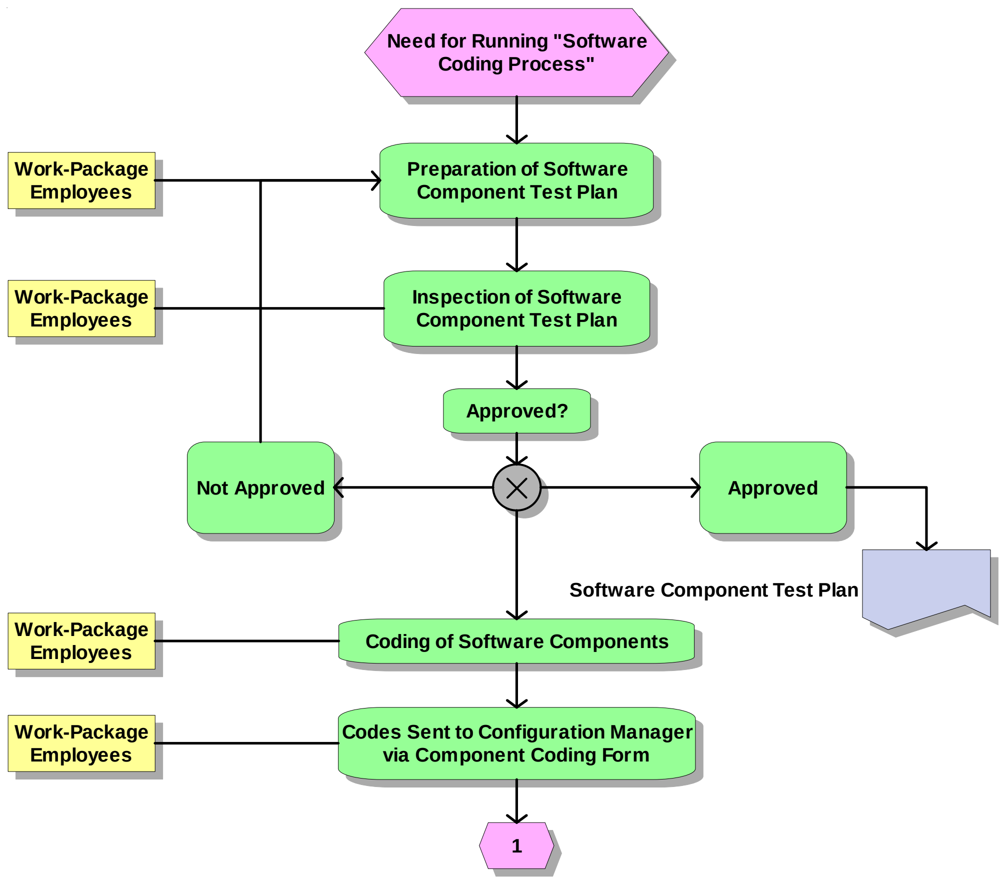

Need for Running "Software Coding Process"
Work-Package Employees
Preparation of Software Component Test Plan
Work-Package Employees
Inspection of Software Component Test Plan
Approved?
Not Approved
Approved
Software Component Test Plan
Work-Package Employees
Coding of Software Components
Work-Package Employees
Codes Sent to Configuration Manager via Component Coding Form
1

Figure 15 - Coding Process

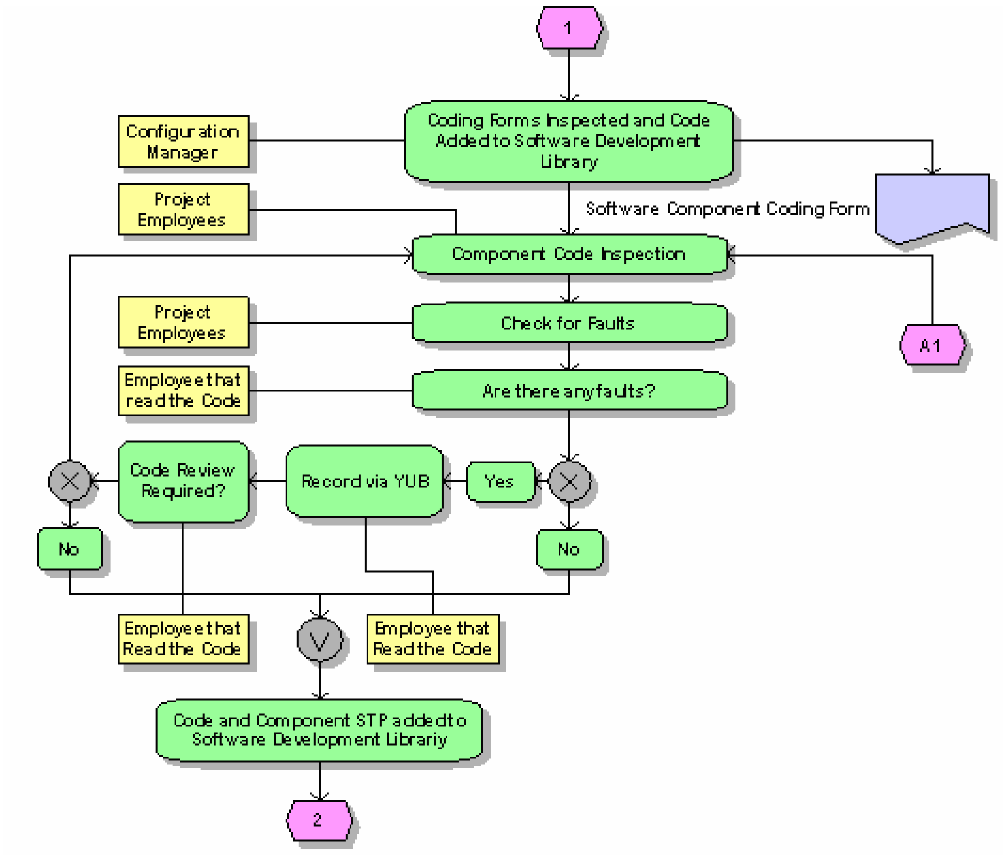

1
Configuration Manager
Coding Forms Inspected and Code Added to Software Development Library
Project Employees
Software Component Coding Form
Component Code Inspection
A1
Project Employees
Check for Faults
Employee that read the Code
Are there any faults?
Code Review Required?
Record via YUB
Yes
No
No
Employee that Read the Code
Employee that Read the Code
Code and Component STP added to Software Development Librariy
2

Figure 15 - Coding Process (Continued)

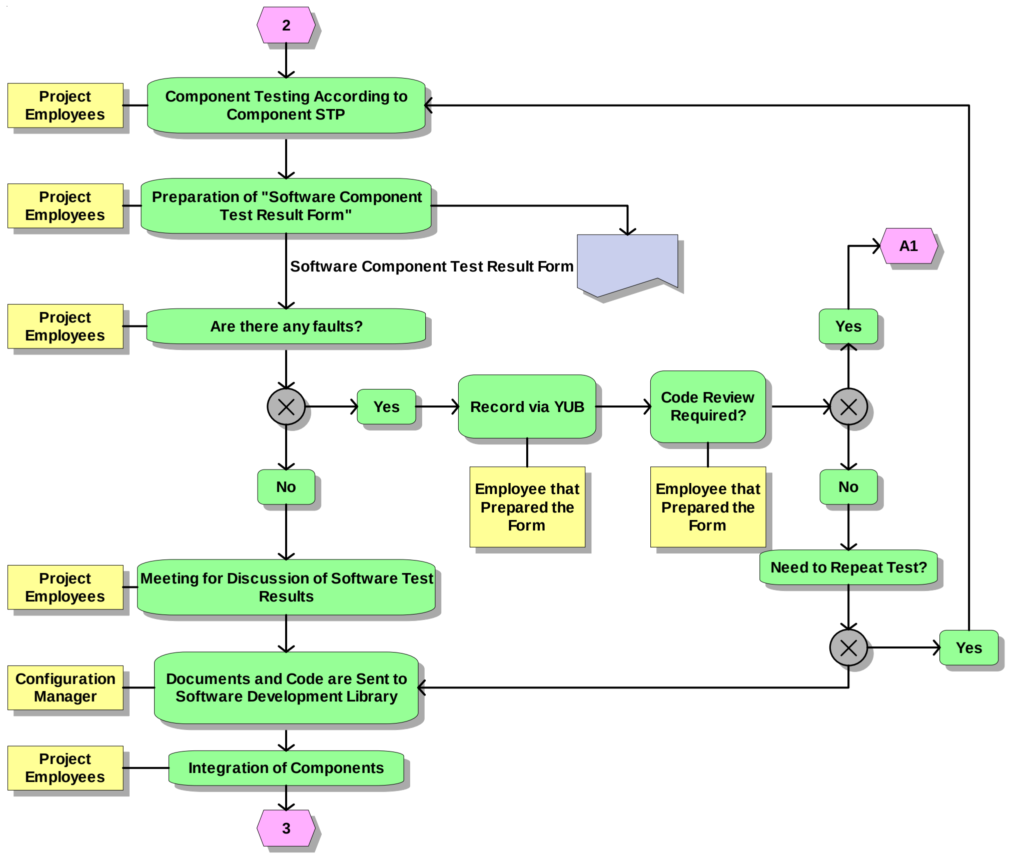

2
Project Employees
Component Testing According to Component STP
Project Employees
Preparation of "Software Component Test Result Form"
Software Component Test Result Form
A1
Project Employees
Are there any faults?
Yes
Yes
Record via YUB
Code Review Required?
No
Employee that Prepared the Form
Employee that Prepared the Form
No
Need to Repeat Test?
Project Employees
Meeting for Discussion of Software Test Results
Yes
Configuration Manager
Documents and Code are Sent to Software Development Library
Project Employees
Integration of Components
3

Figure 15 - Coding Process (Continued)

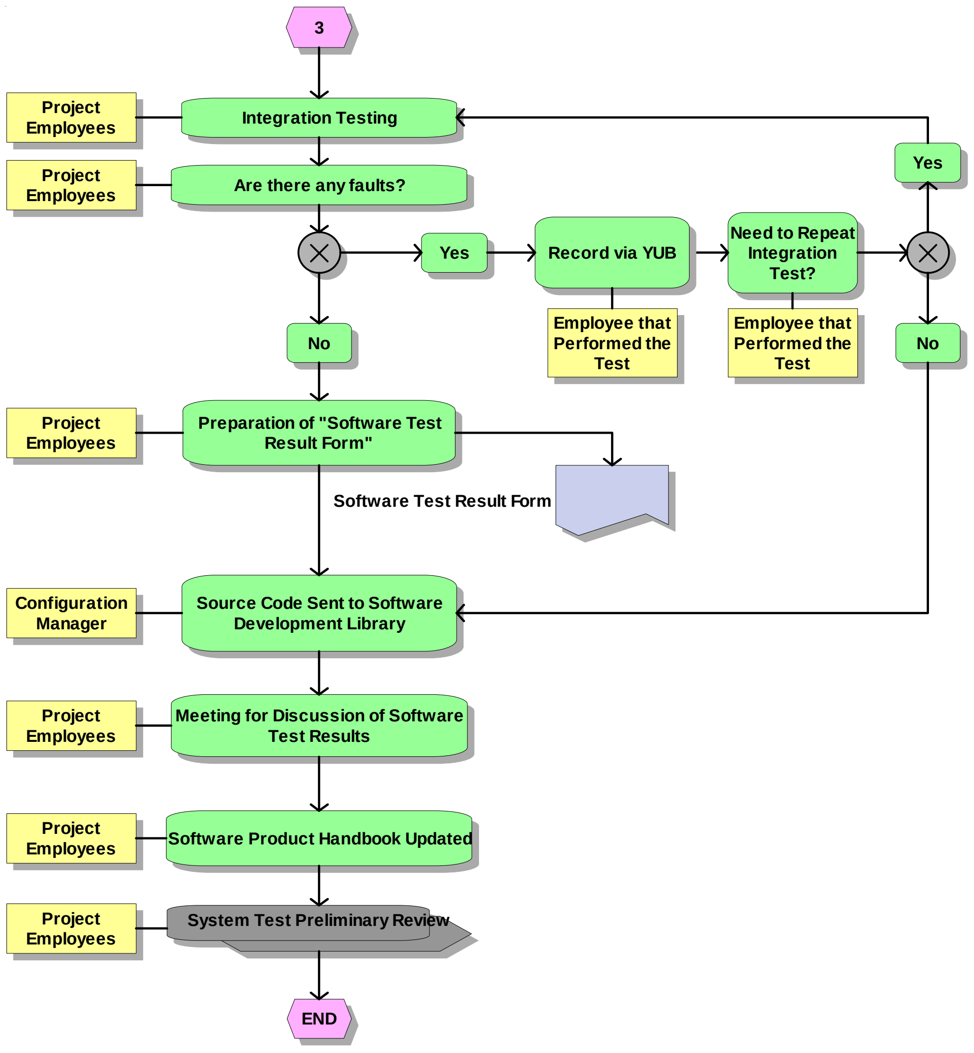
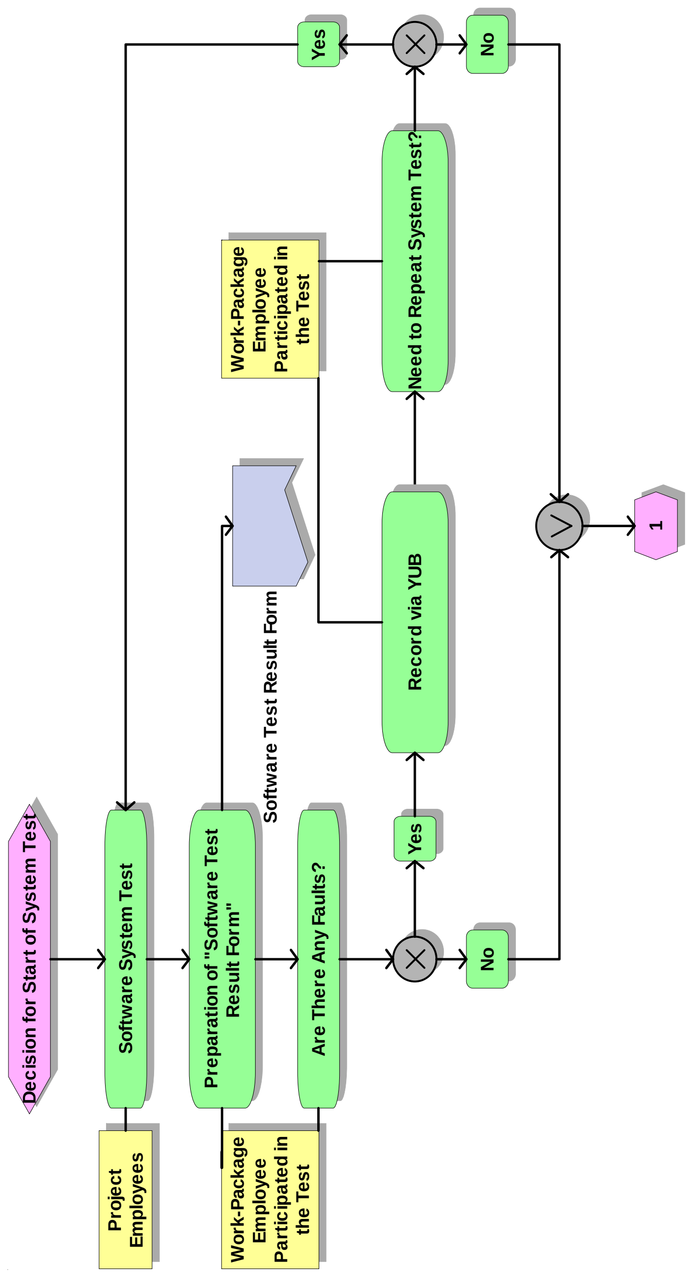

3
Project Employees
Integration Testing
Project Employees
Are there any faults?
Yes
Record via YUB
Employee that Performed the Test
Need to Repeat Integration Test?
Employee that Performed the Test
Yes
No
No
Project Employees
Preparation of "Software Test Result Form"
Software Test Result Form
Configuration Manager
Source Code Sent to Software Development Library
Project Employees
Meeting for Discussion of Software Test Results
Project Employees
Software Product Handbook Updated
Project Employees
System Test Preliminary Review
END

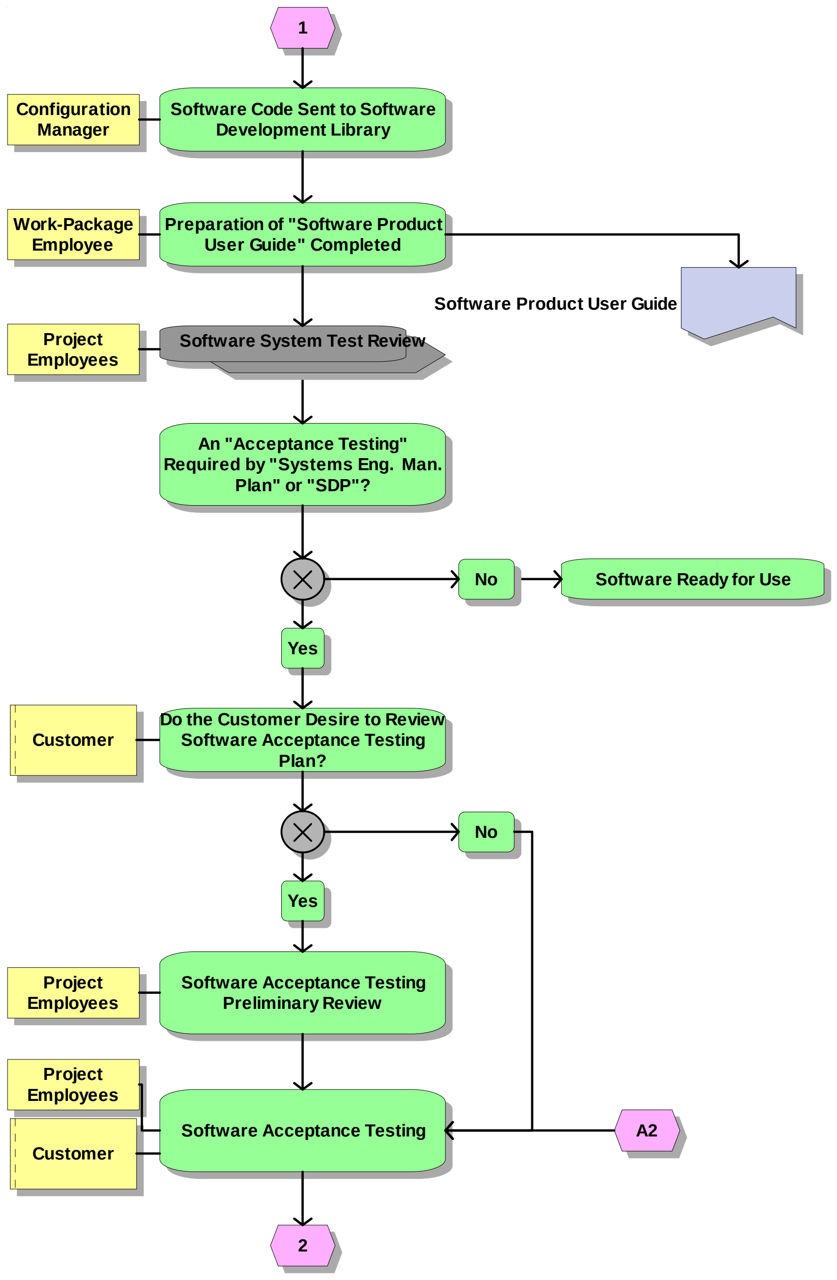
Figure 15 - Coding Process (Continued)

Software Testing Process



Decision for Start of System Test
Project Employees
Software System Test
Work-Package Employee Participated in the Test
Preparation of "Software Test Result Form"
Software Test Result Form
Are There Any Faults?
No
Yes
Record via YUB
Work-Package Employee Participated in the Test
Need to Repeat System Test?
Yes
No
1

Figure 16 - Software Testing Process



1
Configuration Manager
Software Code Sent to Software Development Library
Work-Package Employee
Preparation of "Software Product User Guide" Completed
Software Product User Guide
Project Employees
Software System Test Review
An "Acceptance Testing" Required by "Systems Eng. Man. Plan" or "SDP"?
No
Software Ready for Use
Yes
Customer
Do the Customer Desire to Review Software Acceptance Testing Plan?
No
Yes
Project Employees
Software Acceptance Testing Preliminary Review
Project Employees
Customer
Software Acceptance Testing
A2
2

Figure 16 - Software Testing Process (Continued)

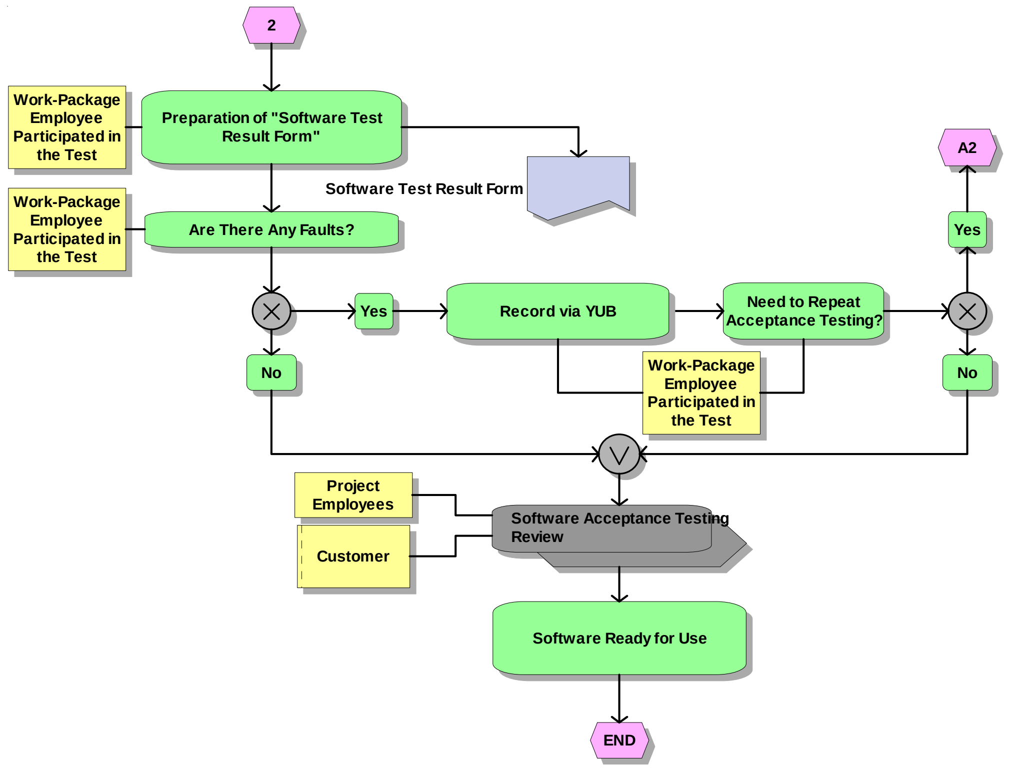

2
Work-Package Employee Participated in the Test
Preparation of "Software Test Result Form"
Software Test Result Form
A2
Work-Package Employee Participated in the Test
Are There Any Faults?
Yes
Yes
Record via YUB
Need to Repeat Acceptance Testing?
No
No
Work-Package Employee Participated in the Test
Project Employees
Software Acceptance Testing Review
Customer
Software Ready for Use
END

Figure 16 - Software Testing Process (Continued)

APPENDIX B

MODIFIED SOFTWARE DEVELOPMENT PROCESS

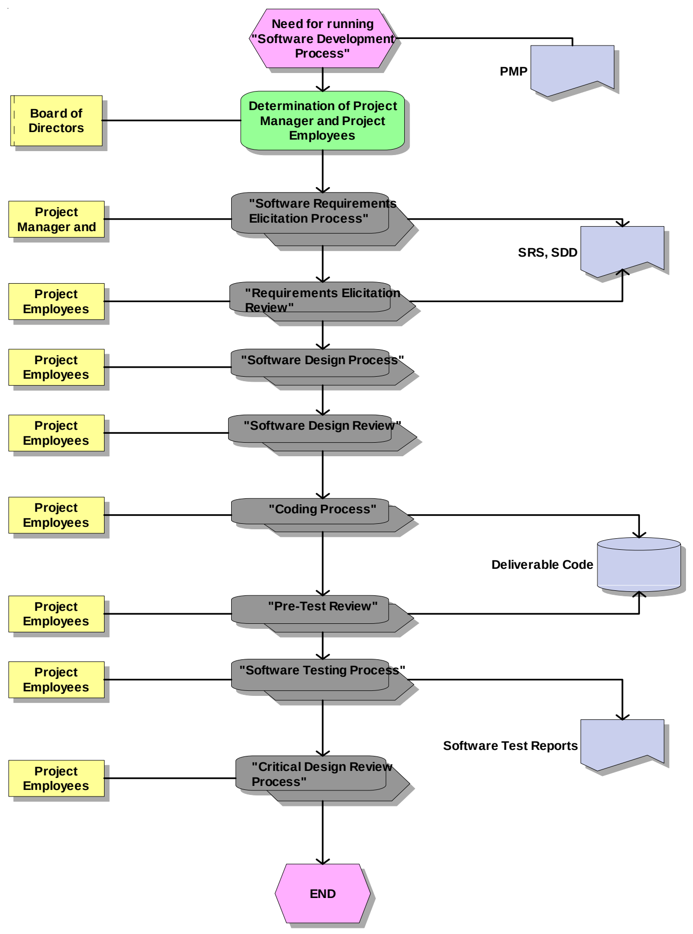


Figure 17 – Modified Software Development Process

Modified Requirements Elicitation Process

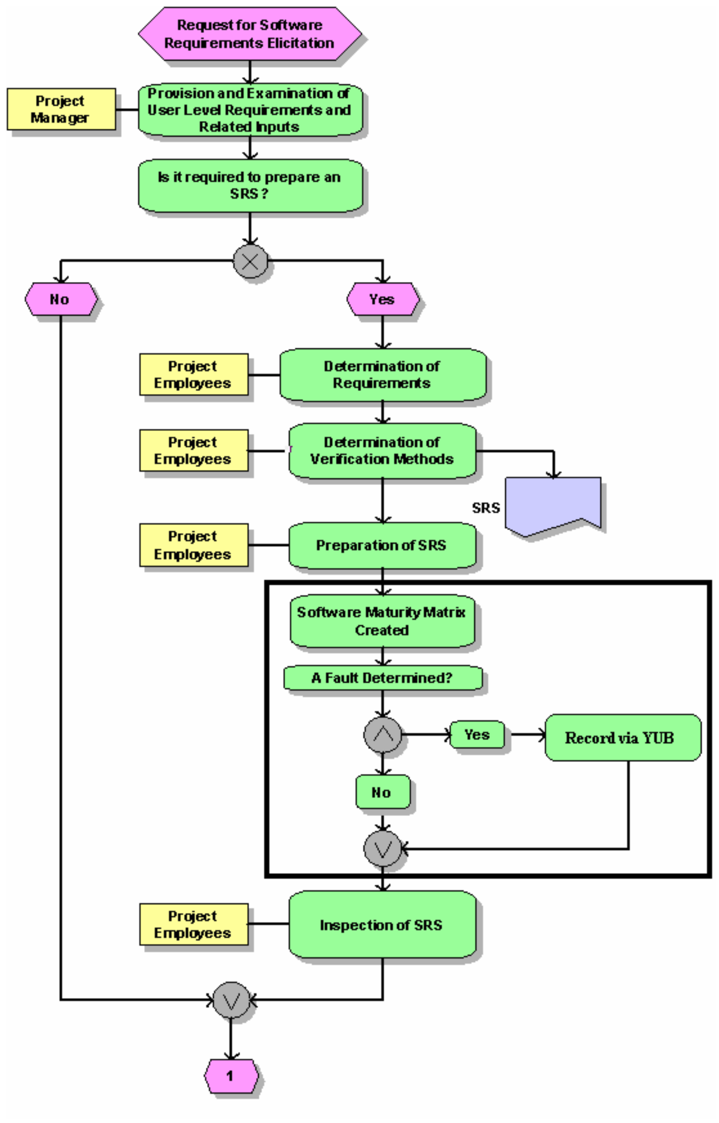

Request for Software Requirements Elicitation
Project Manager
Provision and Examination of User Level Requirements and Related Inputs
Is it required to prepare an SRS ?
No
Yes
Project Employees
Determination of Requirements
Project Employees
Determination of Verification Methods
SRS
Project Employees
Preparation of SRS
Software Maturity Matrix Created
A Fault Determined?
Yes
Record via YUB
No
Project Employees
Inspection of SRS
1

Figure 18 - Modified Requirements Elicitation Process

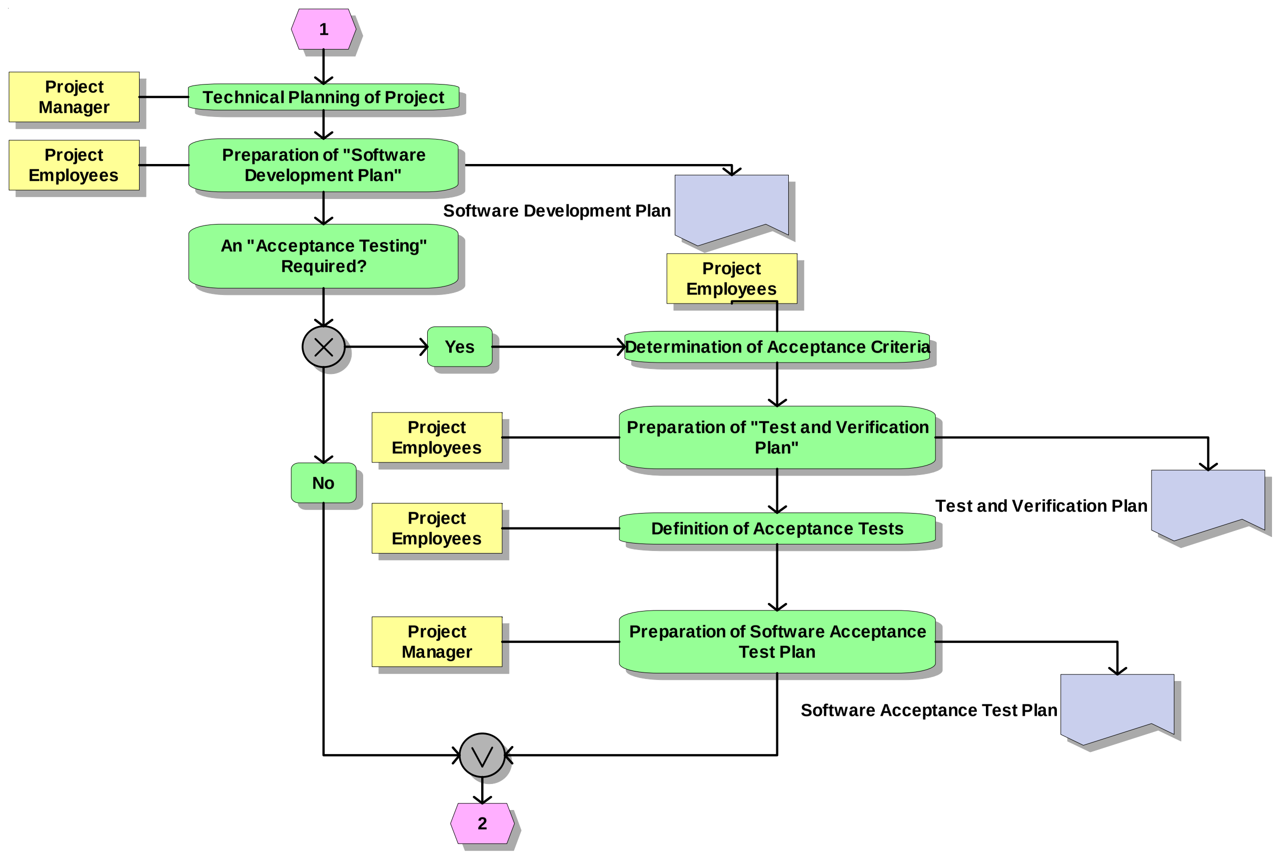
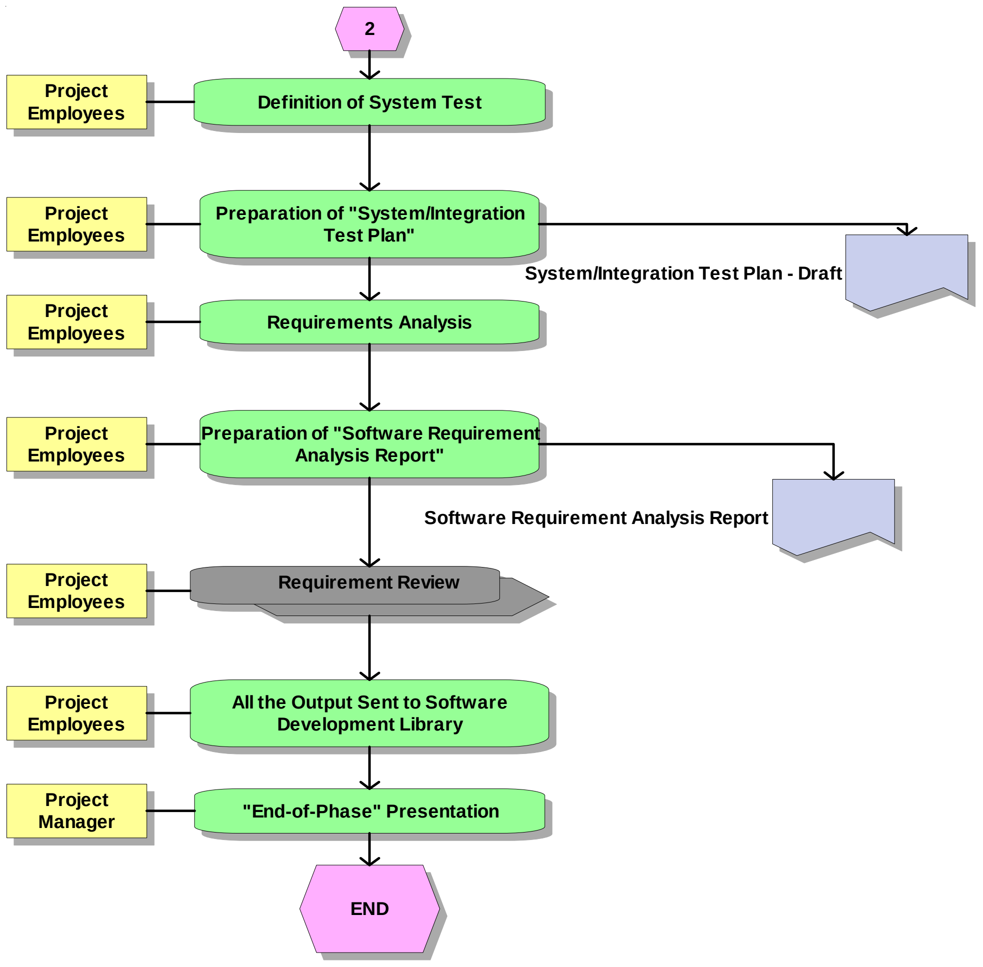

1
Project Manager
Technical Planning of Project
Project Employees
Preparation of "Software Development Plan"
Software Development Plan
An "Acceptance Testing" Required?
Project Employees
Yes
Determination of Acceptance Criteria
Project Employees
Preparation of "Test and Verification Plan"
No
Test and Verification Plan
Project Employees
Definition of Acceptance Tests
Project Manager
Preparation of Software Acceptance Test Plan
Software Acceptance Test Plan
2

Figure 18 - Modified Requirements Elicitation Process (Continued)



2
Project Employees
Definition of System Test
Project Employees
Preparation of "System/Integration Test Plan"
System/Integration Test Plan - Draft
Project Employees
Requirements Analysis
Project Employees
Preparation of "Software Requirement Analysis Report"
Software Requirement Analysis Report
Project Employees
Requirement Review
Project Employees
All the Output Sent to Software Development Library
Project Manager
"End-of-Phase" Presentation
END

Figure 18 - Modified Requirements Elicitation Process (Continued)

Modified Software Design Process

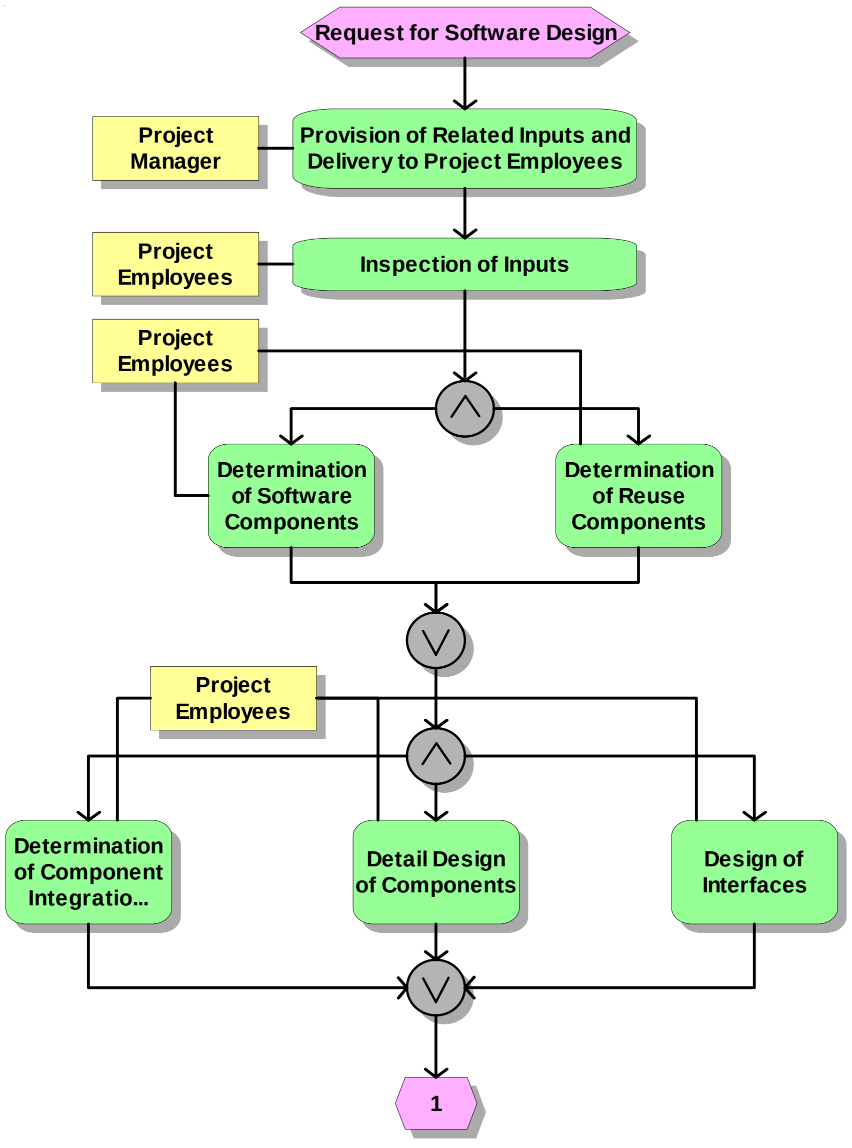

Request for Software Design
Project Manager
Provision of Related Inputs and Delivery to Project Employees
Project Employees
Inspection of Inputs
Project Employees
Determination of Software Components
Determination of Reuse Components
Project Employees
Determination of Component Integratio...
Detail Design of Components
Design of Interfaces
1

Figure 19 – Modified Software Design Process

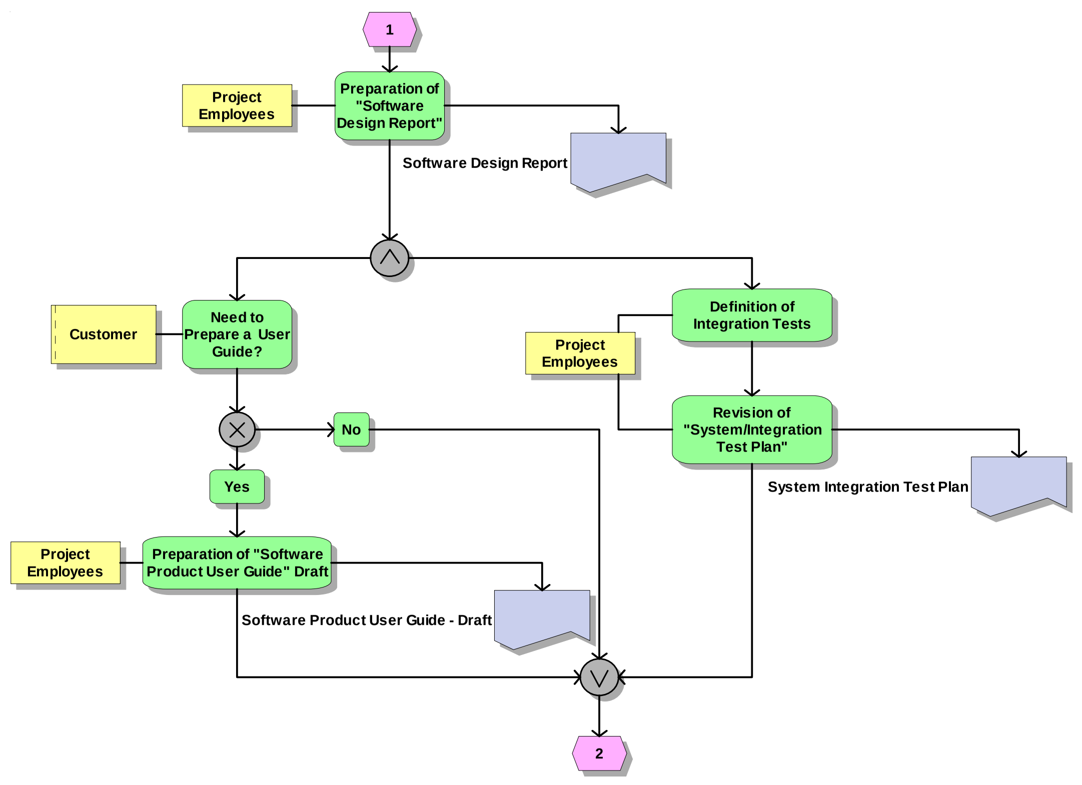

1
Project Employees
Preparation of "Software Design Report"
Software Design Report
Customer
Need to Prepare a User Guide?
No
Yes
Project Employees
Preparation of "Software Product User Guide" Draft
Software Product User Guide - Draft
Definition of Integration Tests
Project Employees
Revision of "System/Integration Test Plan"
System Integration Test Plan
2

Figure 19 – Modified Software Design Process (continued)

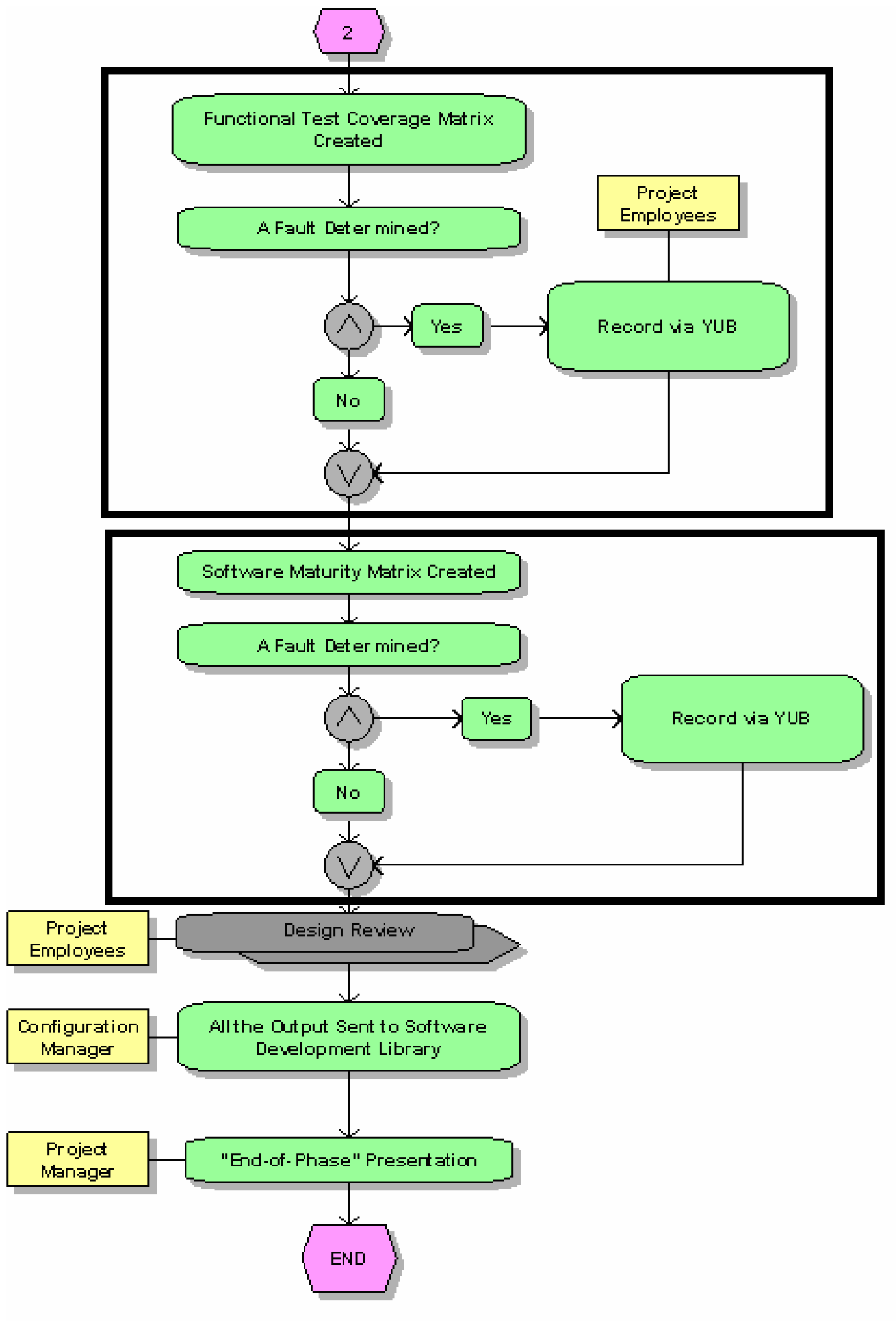


Figure 19 – Modified Software Design Process (continued)

Modified Coding Process

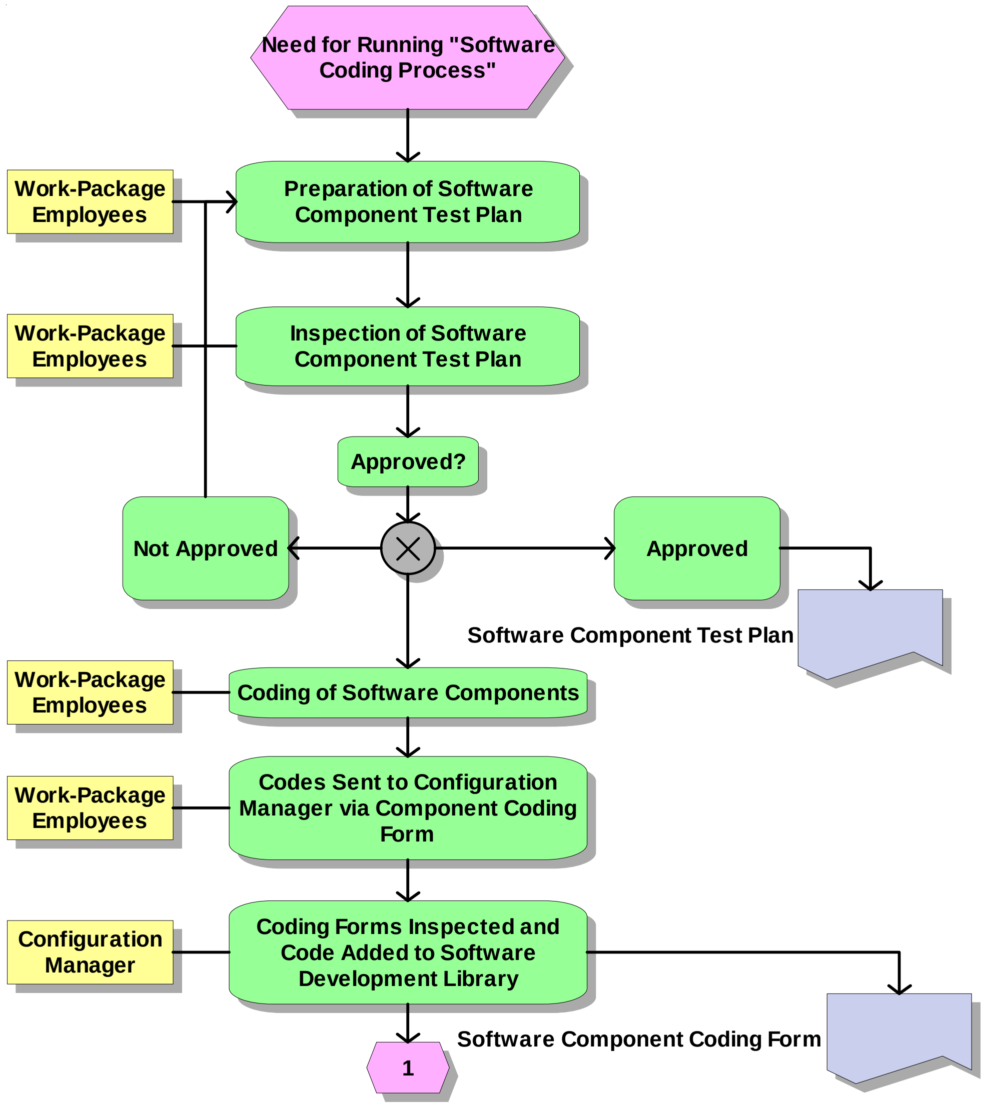

Need for Running "Software Coding Process"
Work-Package Employees
Preparation of Software Component Test Plan
Work-Package Employees
Inspection of Software Component Test Plan
Approved?
Not Approved
Approved
Software Component Test Plan
Work-Package Employees
Coding of Software Components
Work-Package Employees
Codes Sent to Configuration Manager via Component Coding Form
Configuration Manager
Coding Forms Inspected and Code Added to Software Development Library
Software Component Coding Form
1

Figure 20 – Modified Coding Process

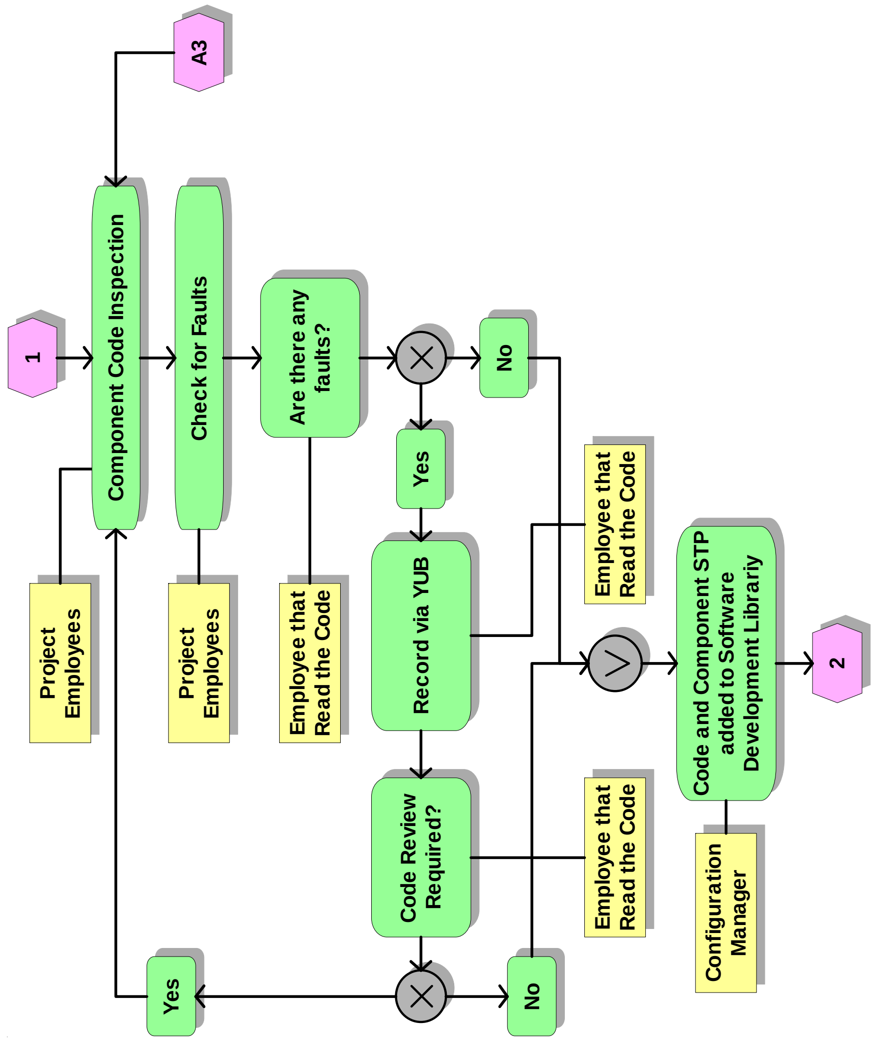

1
A3
Component Code Inspection
Project Employees
Check for Faults
Project Employees
Are there any faults?
Employee that Read the Code
No
Yes
Record via YUB
Employee that Read the Code
Code Review Required?
Employee that Read the Code
Yes
No
Code and Component STP added to Software Development Librariy
Configuration Manager
2

Figure 20 - Modified Coding Process (Continued)

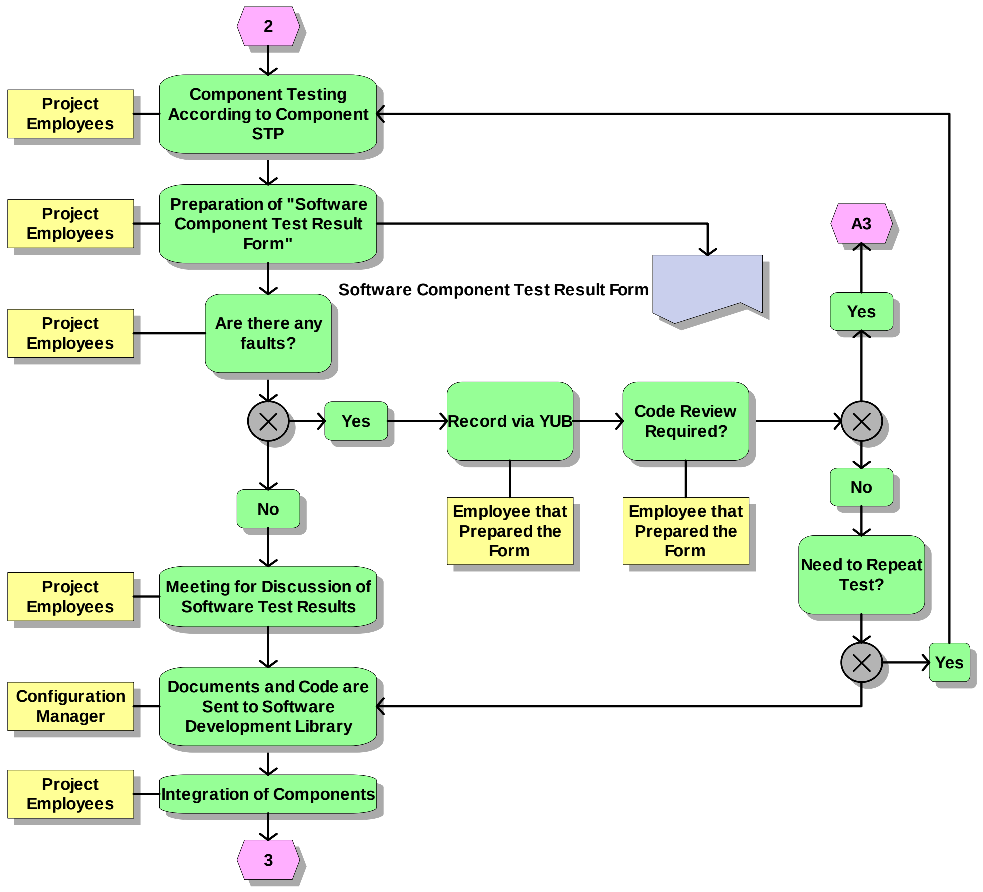

2
Project Employees
Component Testing According to Component STP
Project Employees
Preparation of "Software Component Test Result Form"
Software Component Test Result Form
A3
Project Employees
Are there any faults?
Yes
Yes
Record via YUB
Code Review Required?
No
Employee that Prepared the Form
Employee that Prepared the Form
No
Need to Repeat Test?
Project Employees
Meeting for Discussion of Software Test Results
Yes
Configuration Manager
Documents and Code are Sent to Software Development Library
Project Employees
Integration of Components
3

Figure 20 - Modified Coding Process (Continued)

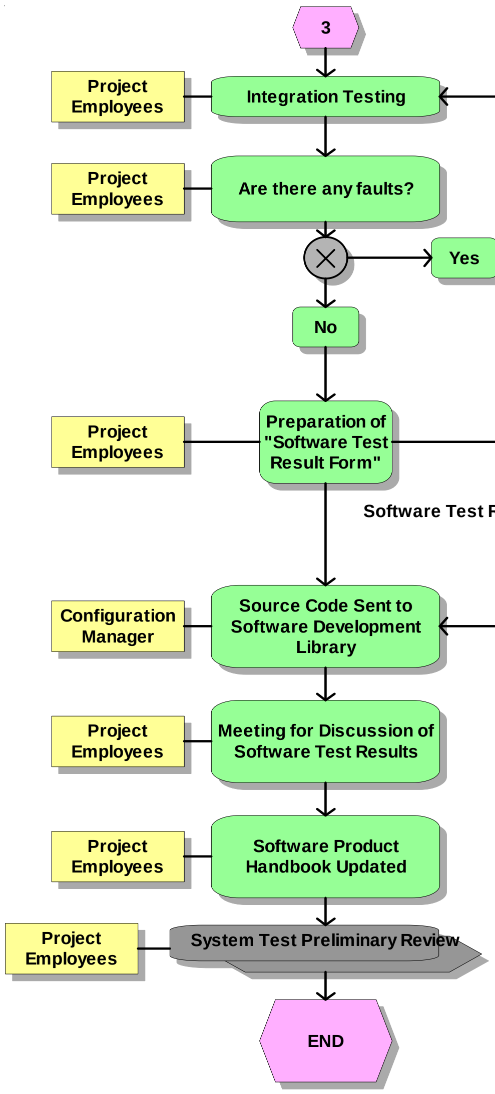

3
Project Employees
Integration Testing
Project Employees
Are there any faults?
Yes
Record via YUB
Need to Repeat Integration Test?
Yes
No
Employee that Performed the Test
Employee that Performed the Test
No
Project Employees
Preparation of "Software Test Result Form"
Software Test Result Form
Configuration Manager
Source Code Sent to Software Development Library
Project Employees
Meeting for Discussion of Software Test Results
Project Employees
Software Product Handbook Updated
Project Employees
System Test Preliminary Review
END

Figure 20 - Modified Coding Process (Continued)

Modified Software Testing Process

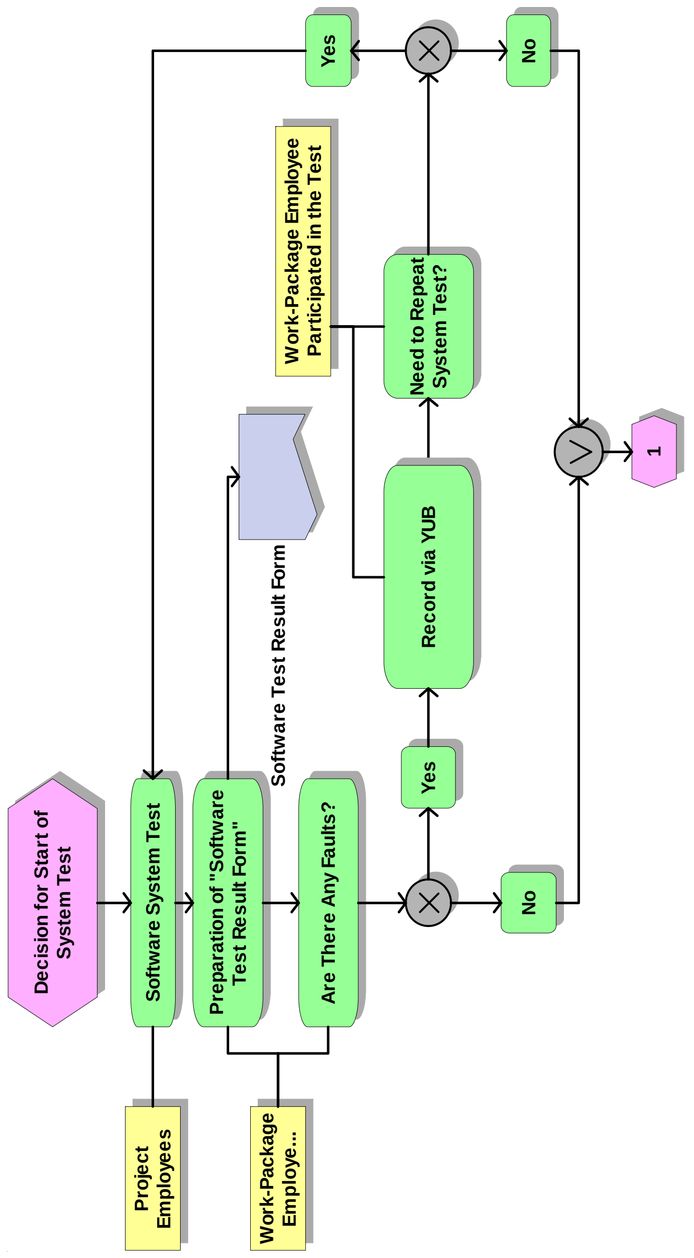

Decision for Start of System Test
Project Employees
Software System Test
Preparation of "Software Test Result Form"
Work-Package Employe...
Are There Any Faults?
Yes
No
Software Test Result Form
Record via YUB
Work-Package Employee Participated in the Test
Need to Repeat System Test?
Yes
No
1

Figure 21 – Modified Software Testing Process

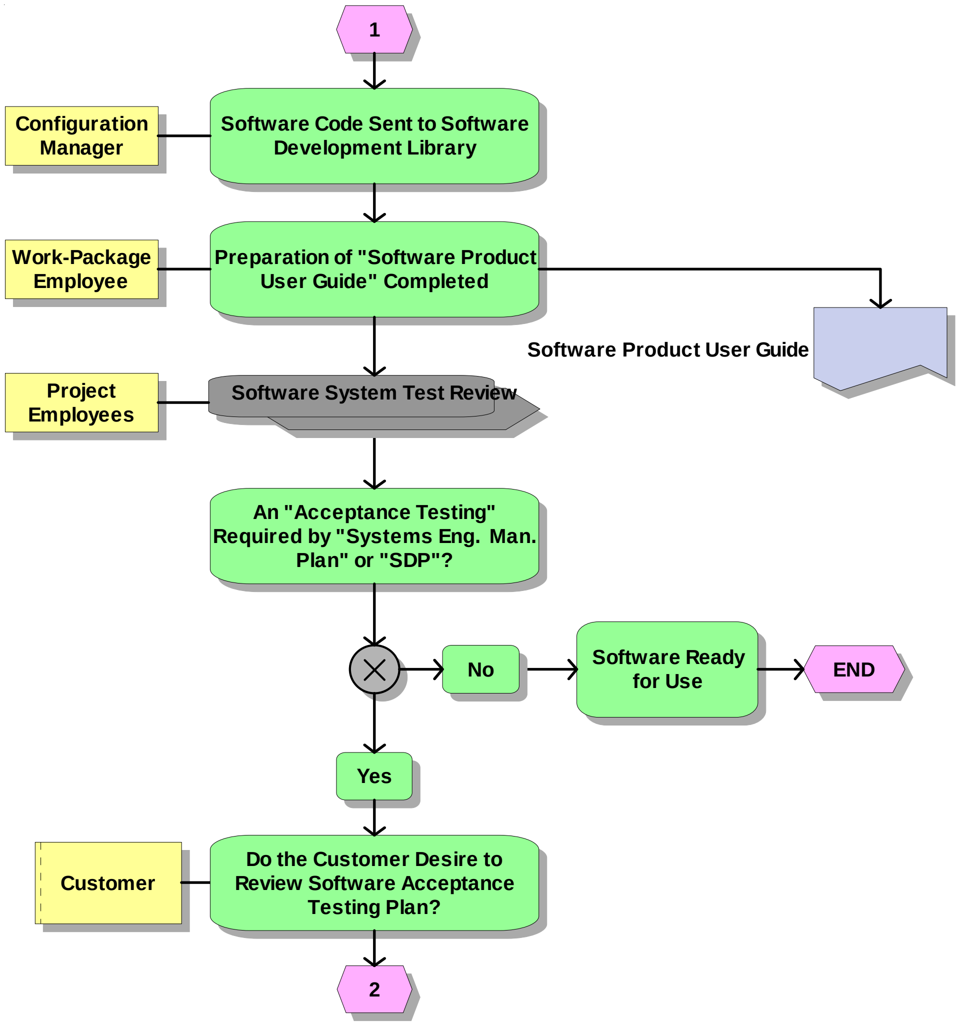

1
Configuration Manager
Software Code Sent to Software Development Library
Work-Package Employee
Preparation of "Software Product User Guide" Completed
Software Product User Guide
Project Employees
Software System Test Review
An "Acceptance Testing" Required by "Systems Eng. Man. Plan" or "SDP"?
No
Software Ready for Use
END
Yes
Customer
Do the Customer Desire to Review Software Acceptance Testing Plan?
2

Figure 21 – Modified Software Testing Process (Continued)

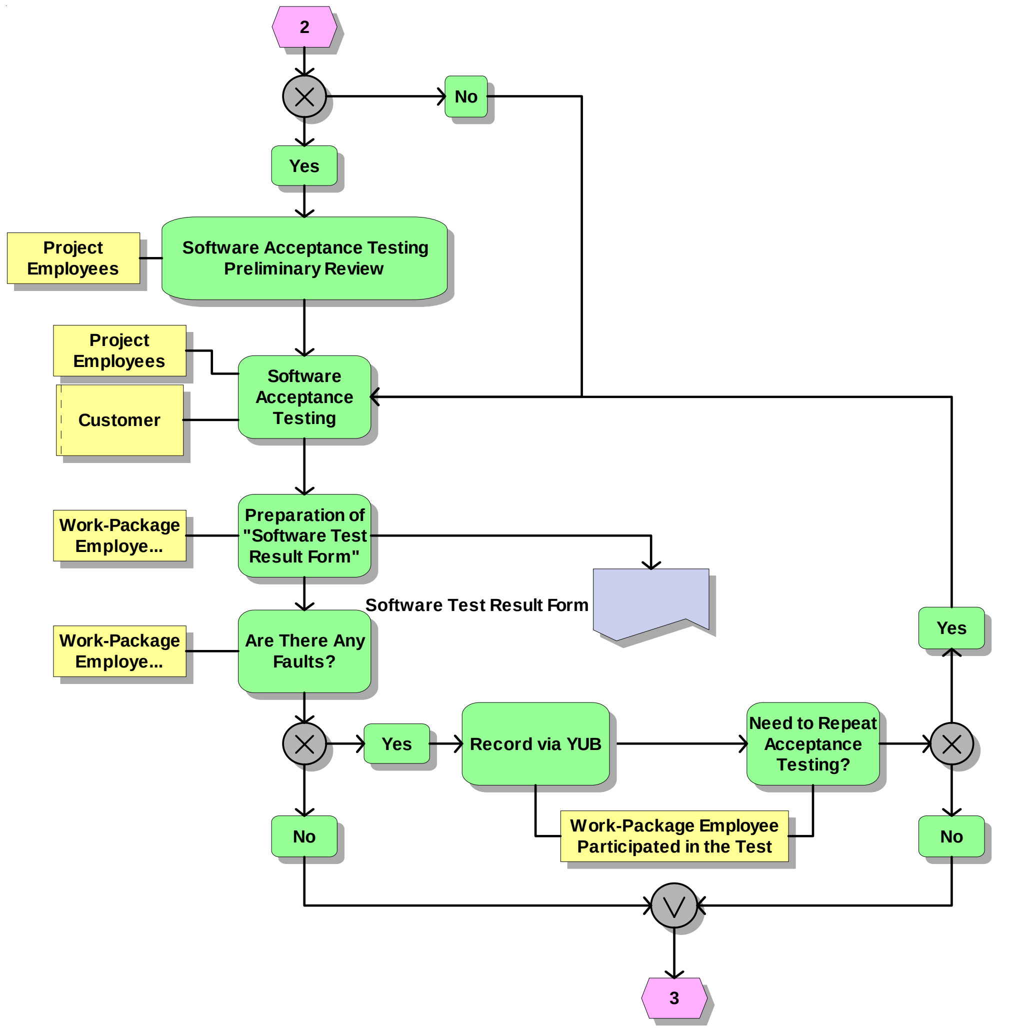

2
No
Yes
Project Employees
Software Acceptance Testing Preliminary Review
Project Employees
Customer
Software Acceptance Testing
Work-Package Employe...
Preparation of "Software Test Result Form"
Software Test Result Form
Work-Package Employe...
Are There Any Faults?
Yes
Yes
Record via YUB
Need to Repeat Acceptance Testing?
No
Work-Package Employee Participated in the Test
No
3

Figure 21 – Modified Software Testing Process (Continued)

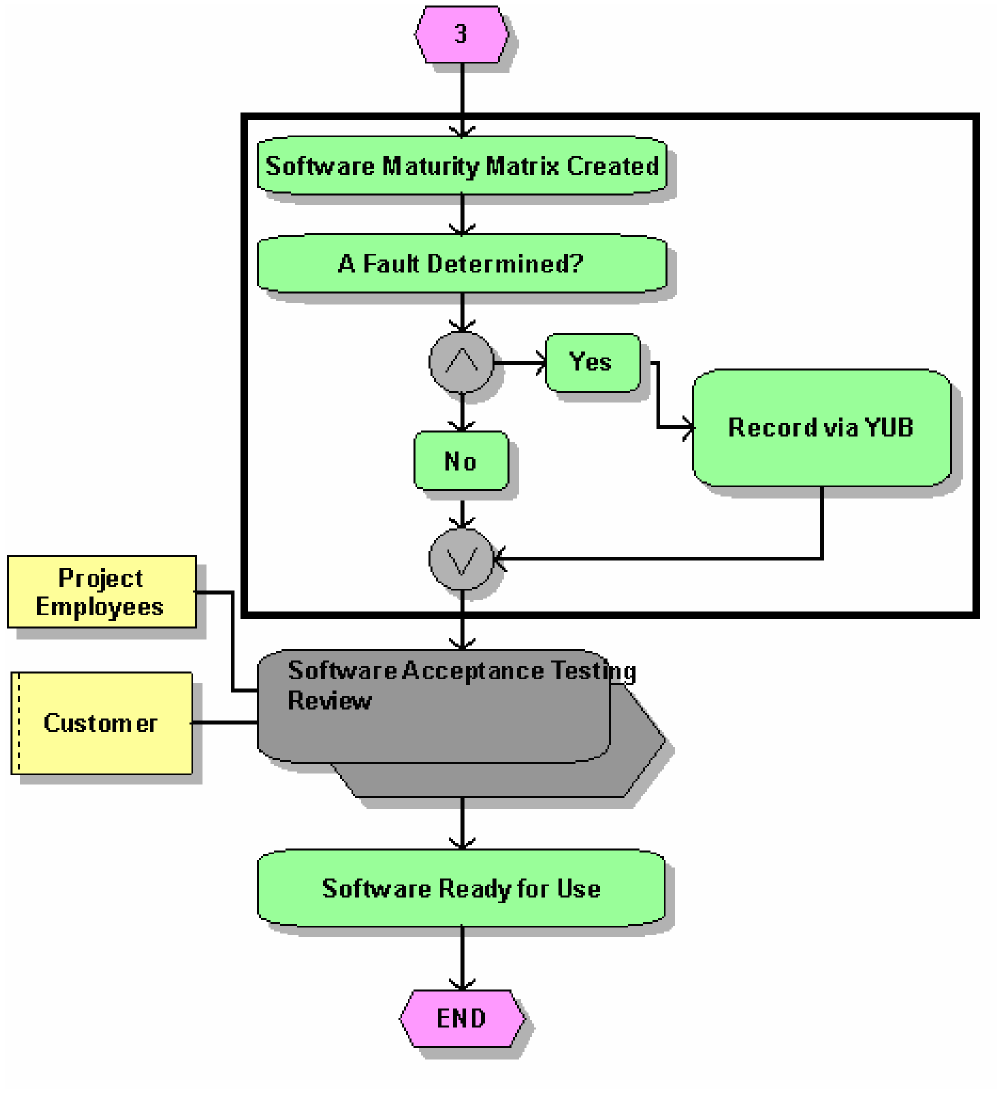


Figure 21 – Modified Software Testing Process (Continued)

APPENDIX C

ORIGINAL PROCESSES STEPCODES

Table 15 - Stepcodes for Original Processes

Stepcode	Development Phase	Step Name
1	Requirements Elicitation	Provision of Related Inputs and Delivery to Project Employees
2	Requirements Elicitation	Preparation of SRS
3	Requirements Elicitation	Technical Planning of the Project
4	Requirements Elicitation	Preparation of Acceptance Testing
5	Requirements Elicitation	Declaration of System Test Requirements
6	Requirements Elicitation	Preparation of Requirements Analysis Report
7	Requirements Elicitation	Requirements Analysis Review
8	Requirements Elicitation	Requirements Elicitation Review
9	Design	Provision of Related Inputs and Delivery to Project Employees

Table 15 - Stepcodes for Original Processes (Continued)

Stepcode	Development Phase	Step Name
10	Design	Inspection of Inputs
11	Design	Determination of Software Components, and Determination of Reuse Components
12	Design	Determination of Component Integration Order
13	Design	Detail Design of Components
14	Design	Design of Interfaces
15	Design	Preparation of Software Design Report
16	Design	Definition of Integration Tests
17	Design	Preparation of System/Integration Test Plan
18	Design	Software Design Review
19	Coding	Preparation of Software Component Test Plan
20	Coding	Inspection of Software Component Test Plan
21	Coding	Coding of Software Components
22	Coding	Inspection of Coding Forms by the Configuration Manager
23	Coding	Component Code Inspection
24	Coding	Component Testing According to Component STP

Table 15 - Stepcodes for Original Processes (Continued)

Stepcode	Development Phase	Step Name
25	Coding	Component Code Inspection
26	Coding	Integration of Components
27	Coding	Integration Testing
28	Coding	New Revision of Software Product Handbook
29	Coding	Pre-Test Review
30	Testing	System Test
31	Testing	Preparation of System Test Result Form
32	Testing	Final Version of Software Product User Guide
33	Testing	Software System Test Review
34	Testing	Software Acceptance Testing Preliminary Review
35	Testing	Software Acceptance Testing
36	Testing	Software Test Result Form
37	Testing	Critical Design Review

APPENDIX D

MODIFIED PROCESSES STEPCODES

Table 16 – Stepcodes for Modified Processes

Stepcode	Development Phase	Step Name
1	Requirements Elicitation	Provision of Related Inputs and Delivery to Project Employees
2	Requirements Elicitation	Preparation of SRS
101	Requirements Elicitation	Preparation of Software Maturity Matrix
3	Requirements Elicitation	Technical Planning of the Project
4	Requirements Elicitation	Preparation of Acceptance Testing
5	Requirements Elicitation	Declaration of System Test Requirements
6	Requirements Elicitation	Preparation of Requirements Analysis Report
7	Requirements Elicitation	Requirements Analysis Review
8	Requirements Elicitation	Requirements Elicitation Review

Table 16 - Stepcodes for Modified Processes (Continued)

Stepcode	Development Phase	Step Name
9	Design	Provision of Related Inputs and Delivery to Project Employees
10	Design	Inspection of Inputs
11	Design	Determination of Software Components, and Determination of Reuse Components
12	Design	Determination of Component Integration Order
13	Design	Detail Design of Components
14	Design	Design of Interfaces
15	Design	Preparation of Software Design Report
16	Design	Definition of Integration Tests
17	Design	Preparation of System/Integration Test Plan
102	Design	Preparation of Functional Test Coverage Matrix
103	Design	Preparation of Software Maturity Matrix
18	Design	Software Design Review
19	Coding	Preparation of Software Component Test Plan
20	Coding	Inspection of Software Component Test Plan
21	Coding	Coding of Software Components

Table 16 - Stepcodes for Modified Processes (Continued)

Stepcode	Development Phase	Step Name
22	Coding	Inspection of Coding Forms by the Configuration Manager
23	Coding	Component Code Inspection
24	Coding	Component Testing According to Component STP
25	Coding	Component Code Inspection
26	Coding	Integration of Components
27	Coding	Integration Testing
28	Coding	New Revision of Software Product Handbook
29	Coding	Pre-Test Review
30	Testing	System Test
31	Testing	Preparation of System Test Result Form
32	Testing	Final Version of Software Product User Guide
33	Testing	Software System Test Review
34	Testing	Software Acceptance Testing Preliminary Review
35	Testing	Software Acceptance Testing
36	Testing	Software Test Result Form

Table 16 - Stepcodes for Modified Processes (Continued)

Stepcode	Development Phase	Step Name
104	Testing	Preparation of Software Maturity Matrix
37	Testing	Critical Design Review

APPENDIX E

SIMULATION RESULTS FOR PROJECT 1

In this appendix, for the sake of brevity, only last ten rows of summary of results of P1 has been included. The full results have been presented in the attached CD, under the folder named "Simulation_Results". Results of the projects are available in the folders bearing the names of respective projects.

The simulation tool is run 83 times for P1, 62 times for P2, 90 times for P3, 77 times for P4, 92 times for P5, 104 times for P6, 231 times for P7, 72 times for P8, and 82 times for P9, adding up to 893.

Table 17 - Simulation Results for Project 1

Original Processes					TTC	NoFI	NoFD	KLOC	Modified Processes									TTC	NoFI	NoFD	KLOC	MFD	MDD	MRT	MTbFD	MTbFR
Run	TTC	NoFI	NoFD	KLOC	n'	n'	n'	n'	TTC	NoFI	NoFD	KLOC	MFD	MDD	MRT	MTbFD	MTbFR	n'	n'	n'	n'	n'	n'	n'	n'	n'
74	323	203	141	14	6	39	64	56	321	209	163	15	119	172	11	53	64	12	50	83	78	26	22	15	64	47
75	323	200	139	14	6	40	64	56	310	183	134	13	112	169	10	57	67	12	49	82	77	26	22	15	63	47
76	299	169	114	11	6	40	64	57	324	206	137	15	121	178	11	57	68	12	49	81	77	26	22	15	63	47
77	343	211	127	15	7	41	63	59	319	213	148	14	114	177	10	63	73	12	49	80	76	25	21	15	62	46
78	349	230	163	16	8	45	70	64	322	206	159	15	117	178	10	61	71	12	49	80	76	25	21	15	61	45
79	313	192	131	13	8	45	70	63	353	243	160	17	132	189	10	57	67	13	53	80	79	26	21	15	61	45
80	323	194	132	14	8	44	69	63	298	165	109	11	104	165	10	61	71	13	54	83	81	27	21	15	60	44
81	298	161	100	11	8	45	71	64	306	184	132	13	113	168	11	55	66	13	53	83	81	26	21	15	60	44
82	313	193	135	13	8	45	71	63	330	206	151	15	124	170	10	46	56	13	53	82	80	26	21	15	64	47
83	312	183	110	13	8	44	71	62	321	198	139	15	119	187	10	68	78	13	52	81	80	26	21	15	64	47
	Mean Values								Mean Values																	
	313	184	123	13					316	194	141	14	116	176	11	55	64									