www.ijecs.in
International Journal Of Engineering And Computer Science ISSN:2319-7242
Volume 2 Issue 4 April, 2013 Page No. 1147-1155

Software Quality Prediction: A Review and Current Trends

P. Ranjeet Kumar *, R. Ramesh**, T.Venkat Narayana Rao ***, Shireesha Dara****

* Student B.Tech, Third Year, C.S.E, KKR & KSR Institute of Technology and Sciences [KITS], Vinjanampadu, Guntur., A.P, INDIA
**Professor, Computer Science and Engineering, KKR & KSR Institute of Technology and Sciences [KITS], Vinjanampadu, Guntur. A.P, INDIA
***Professor, Computer Science and Engineering, Guru Nanak Institutions Technical Campus
****Asst. Professor, Computer Science and Engineering, Guru Nanak Institutions Technical Campus
R.R. District, Ibrahimpatnam, A.P, INDIA, tvnrbobby@yahoo.com

Abstract:
Software Engineering is the area to analysis, design, development and maintenance of software. Quality is the main constraint about the success of the software design. Works on testability of components or component-based software have proposed several techniques for increasing testability of component-based software systems. This work aims at reviewing these techniques for understanding their similarities and differences. This helps in evaluating proposed techniques as per their contribution in solving the concerned problems. The major quality attribute of a software product is the degree to which it can be relied upon to perform its intended function. Evaluation, prediction, and improvement of this attribute have been of concern to designers and users of computers and software from the early days of their evolution. Idea behind code reuse is that a partial or complete computer program written at one time can be written into another program at later time. Simulators also calculate the mean relative error between original effort and calculated efforts. Selecting the right software tool is one of the most important decisions taken by company; the success of the company will depend predominantly on it. Moreover, it is hard to recognize which tool is most appropriate technology transfer is an issue of major significance for organisations wishing to use reuse technology. This paper gives overview about the software models and reuse technology of software and the trends, which are currently used.
.

Keywords: Software design, software reliability, program, QC, software development tools

I. INTRODUCTION

Software engineering is the profession to analysis, design, development and maintenance of software. Software engineering mainly focuses on management process of software development and expenditure representations with cost effective software development [2]. Quality of the software can be improved by the fault prediction in software using the different metrics like size or performance or complexity or combination of these. These metrics are evaluated to predict the fault to produce the quality in software. These problems require better testing and quality assurance techniques. Testability of a software system is an effective and viable technique of reducing the testing cost, and, increasing testing effectiveness. Testability is not only the indicator of testing effectiveness, but, also, a measurable indicator of quality of a software development process. Testing involves test-case generation, test-case execution and test evaluation. All the aspects of software development that ease these activities directly or indirectly make a software system more testable. The role of computer software has undergone significant change over a time span of little more than 50 years. Today, software takes on a dual role. It is a product and, at the same time, the vehicle for delivering a product. As a product, it delivers the computing potential embodied by computer hardware or, more broadly, a network of computers that are accessible by local hardware. Whether it resides within a cellular phone or operates inside a mainframe computer, software is information transformer, producing, managing, acquiring, modifying, displaying, or transmitting information that can be as simple as a single bit or as complex as a multimedia presentation. The primary grounds influencing these issues are either requirements of company or goals set by the company [1].

Thus it is important to get a tool which can be configured with needs of users and adaptable with newer technologies. The portability of tool also plays a crucial role in selecting a tool Software reuse exploits a collaboration process in which designers working on new problems can take

advantage of the work of designers who have encountered similar problems in the past. Not only technical problems but also cognitive and social factors inhibit the widespread success of systematic software reuse. An important paradigm shift is to reconceptualize reuse as a collaborative process, in which software designers should not only take advantage of existing reuse repositories, but, through their own work, modify components and evolve reuse repositories.

II. REVIEW OF LITERATURE

Cagatay Catal (2010) gives the detailed literature review and current trends in software fault prediction. The problem occurs to predict the software fault then the previous software fault data was missing and if it may be the new project. To overcome this problem X-means clustering based method used, also fuzzy clustering and k-means clustering-based methods used in the experiments. But the system effecting cost of metrics collection tools are expensive, so need influential model to predict fault with limit fault data. Edward Allen (1998) describes the reliability enhancement of targeting software modules based on genetic programming (GP). In this paper case study, using CBR predicted the quality of Command, Control, and Communications written in Ada. This system provides high accuracy and useful.

A) Testability Techniques for CBS Systems

Researchers have been concerned with testability in order to make software testing more efficient. CBS development aims at maximising reuse of COTS and in house components. Widespread use of COTS and in-house components makes testability more crucial and relevant to software development. This explains why most of the techniques addressing CBS testability are concerned with component testability. Gross etal (2005) proposed the use of models for generating built-in-test for automation and effort reduction[3][4].

B) Problems Addressed By CBS Testability

In order the understand CBS testability, we will try to concentrate on the problems of concern of various testability techniques. This gives a ground for categorising the proposed testability techniques. Further, this lets us understand the contribution of different techniques more explicitly. What are the problems, which are being considered under the purview of CBS testability? Below, we classify the testability techniques as per nature of the addressed problems. This would let us evaluate testability techniques in a realistic manner. Classification is not disjoint; rather, it has to do with major problems that have been considered by researchers. Following are the major problems that have been considered under purview of CBS testability[5].

i) Modeling Component testability

These studies considered the aspects related to the form of a testable component and the way testability can beanalysed and measured. Freedman (1991) showed that it is essential to recognize hidden inputs and outputs [1]. Further, he stressed on the fact that we should specify output domain so that it matches the set of outputs caused by input values. This is because if we don't know what are inputs and outputs, then, it would be impossible to evaluate the result of test case execution. It may not be possible to specify outputs according to exact set of outputs obtained by executing the set of inputs in every case. We should be able to analyze the testability on the basis of its code and internal structure. This is important because certain decision regarding testability can be made only after the analysis of its structure or code. A work entitled "Testability Analysis for Software Components" aims at testability analysis of C or Ada component to recognize those control flows which can be used to test a part of code more easily. A qualitative model incorporating the factors that affect component testability was proposed by Gao and Shih (2005) in a work entitled "A Component Testability Model for Verification and Measurement".

ii) Facilitating User-Oriented Component Testing

Validating the COTS components as per an application context is a key step in successfully developing the CBS systems .A work entitled "Constructing Self-Testable Software Components" aims at providing built in test capability incorporating generation of test cases, implementation of assertions relative to class invariants and class methods pre and post conditions, test driver generation, test retrieval, test history creation and aintenance. Work entitled "Contract-based It integrates STECC method and metamorphic testing to accomplish the specified features. Next, a work tries to capture the dependency between two components. It devises a method of providing dependency information in form of metadata which can be used for generating test cases when integrating components. It also aims at increasing observability based observation points. This helps in modulating the separate concerns, and, facilitates maintenance and evolution of software systems.

iii) Test Support and Automation at Architecture Level

A work entitled "A study on Design for testability in Component-Based Embedded Software" is concerned with test support at architecture level. It identified that techniques for control of messaging, simulation of stubs and deployment environment, testing support in form of built-in test, trace support and support for ad-hoc testing support are required at architectural level[6].

iv) Deployment and run-time test support

Run-time testing is a viable option for integrating components that cannot be tested during traditional integration time testing. Performing tests during deployment or in-service time introduces interference problems, such as undesired side effect in the state of a system or the outside of the system. Major issue with runtime testability is devising models for analyzing interference, and, come up with built-in test support and test infrastructure support to

deal with the interference. A qualitative model was proposed by the work entitled "A Model for the Measurement of the Runtime Testability of Component-based Systems".

III. DISCUSSIONS

Testability techniques for software systems aim at devising methods and guidelines, setting standards, and analyzing artifacts so as to make testing easy. Many testability techniques have been proposed in connection with issues that can make testing easier. We will discuss the various challenges and problems falling in the categories presented in previous section.

A) Challenges and Problems in Modeling Component Testability

What is a testable component? Jerry Gao has identified factors affecting component testability. According to him component testability depends on understadability, observability, component traceability and test support capability of a component .Another challenge is to provide well defined guide lines to incorporate the above mentioned attributes[7].

B) Challenges and Problems in Facilitating User-Oriented Testing

User-oriented testing has to be accomplished without access to source code. This means that user needs to stick to some form of specification based testing. If user wants code based testing, then, vendor of component needs to supply those test cases along with the component. Further, component should facilitate test execution and evaluation. This is supported by self-testable component. A self-testable component requires Built-in-test capabilities, test case set or some specification for creating test case set and infrastructure supporting self-testability. Problem with this approach is that user can not choose test-criteria at his discretion. It is prefixed by the vendor of a component.

C) Challenges and Problems for Test and Automation Support at Architecture Level

Testability is an important concern at architectural level. Test implementation, control of messaging, simulation of stubs and execution environment and built-in test support from components are challenges at architecture level [9]. What are effective ways to accomplish the above mentioned tasks? How can we support the testing of non-functional requirements at architectural level? It has been argued that COTS components must be verified early in the software development life cycle for support of functional and nonfunctional requirements.

D) Challenges and Problems for Run-Time Testing

Challenges in this category comprised of understanding and separating the requirements according to type of affect they produce on a running system. Next, each of the requirements has to be analysed for the possible test support required. In order to meet these challenges, we need to develop models and techniques to understand the way the components interact with each other, and, the affect functionalities to be tested have on functionalities of other components [10].

E) Challenges and problems in measuring component testability

Since, testability may be related to almost all the activities of the software development life cycle, a testability technique needs to be evaluated in terms goodness of solution of the practical problems it helps in solving so as to make testing easier. This implies that we cannot compare testability gain due to a specification standard with that of testability gain resulting from coding standard. This is because we do not have so much refined understanding of Testability gain obtained by following these two standards. Both techniques help in making testing easier in different ways.

IV.SOFTWARE AND HARDWARE RELIABILITY

However, the concepts and theories developed for software reliability could really be applied to any design activity, including hardware design. Once a software (design) defect is properly fixed, it is in general fixed for all time. Although manufacturing can affect the quality of physical components, there application process for software (design) is trivial and can be performed to very high standards of quality. Since introduction and removal of design faults occur during software development, software reliability may be expected to vary during this period [8]. The "design reliability" concept has not been applied to hardware to any extent. It was possible to keep hardly generally less complex logically then software.

A) Software Reliability

Software Reliability is a subfield of software engineering in which practitioners are concerned with measuring and managing software quality. This valuable discipline has inherited much of its theory from hardware reliability and has gained mixed acceptance from the software community[1]. Software Reliability has been regarded as one of the important quality attributes because low reliable software systems have high possibilities of causing serious problems such as the loss of human life, catastrophic mission failures, and the waste of valuable resource investments. Software Reliability is defined as: the probability of failure-free software operation for a specified period of time in a specified environment. It is evident from the definition that there are four key elements associated with the reliability namely element of probability, function of the product, environmental conditions, and time. Software

B) Hardware Reliability

Hardware Reliability is concerned with the random occurrences of undesirable events, or failures, during the life cycle of a physical system. Since a failure phenomenon can only be described in probability terms, the definition of reliability depends heavily on probability concepts. The reliability of a system is defined as the probability that the

system will adequately perform its intended function for a specified interval of time under stated environment conditions. Reliability evaluation using probability methods provides a quantitative measure of system performance. Hence it allows comparison between systems or provides a logical basis for reliability improvement in a system[11] .

C) Modeling

A model is a simple representation of the system or real process comportment or structure. The goal of the modeling process is to reproduce the fundamental relationships in order to realize their clear understanding.

Ideally, these models provide a means of characterizing the development process and enable software reliability practitioners to make predictions about the expected future reliability of software under development. Such techniques allow managers to accurately allocate time, money, and human resources to a project, and assess when a piece of software has reached a point where it can be released with some level of confidence in its reliability.

Characteristics of a Software Reliability Model

A good model presents following properties.

a)It should provide good prediction of future behavior.

b) It should compute useful quantities.

c) It should be simple.

d) It should be widely applicable.

e) It should be based on sound assumptions.

The software reliability modeling is a functional representation of the debated system and the better model offers a viable mechanism for reliability estimation.

ii) Analytical Models

A number of analytical models have been proposed to address the problem of software reliability measurement. These approaches are based mainly on the failure history of software and can be classified according to the nature of the failure process studied as indicated below.

A)Times between Failures Models

In this class of models, the process under study is the time between failures. The most common approach is to assume that the time between, say, the and the failures, follows a distribution whose parameters depend on the number of faults remaining in the program during this interval. Another approach is to treat the failure times as realizations of a stochastic process and use an appropriate time-series model to describe the underlying failure process.

b) Failure Count Models

The interest of this class of models is in the number of faults or failures in specified time intervals rather than in times between failures. The failure counts are assumed to follow a known stochastic process with a time dependent discrete or continuous failure rate. Parameters of the failure rate can be estimated from the observed values of failure counts or from failure times. Estimates of software reliability mean time to next failure, etc., can again be obtained from the relevant equations[8].

c) Fault Seeding Models

The basic approach in this class of models is to "seed" a known number of faults in a program which is assumed to have an unknown number of indigenous faults. The program is tested and the observed numbers of seeded and indigenous faults are counted. From these, an estimate of the fault content of the program prior to seeding is obtained and used to assess software reliability and other relevant measures.

d) Input Domain Based Models

The basic approach taken here is to generate a set of test cases from an input distribution which is assumed to be representative of the operational usage of the program. Because of the difficulty in obtaining this distribution, the input domain is partitioned into a set of equivalence classes, each of which is usually associated with a program path. An estimate of program reliability is obtained from the failures observed during physical or symbolic execution of the test cases sampled from the input domain

D) Reliability of Open Source Technology

Open source technology has gained a significant amount of mind share and has been the subject of much debate. Often promoted as being better than proprietary software (from an ethical and social point of view), and criticized as being unrealistic or too idealistic. The concept itself is based on the philosophy of free software, which advocates freely available source code as a fundamental right. However, open source extends this ideology slightly to present a more commercial approach that includes both a business model and development methodology.

E)Error Estimation For Open Source Software

Many models have been proposed to assess whether a software-testing objective has been met to determine when to stop testing. These models are based on various sets of assumptions about the software and its execution environment. Software reliability growth models (SRGM`s) use data about the times that failures occur to estimate the number of remaining failures in a system .Generally, SRGM`s estimate the number of failures in a system.

V.SOFTWARE PROCUREMENT PROCESS

Some investigation intended to improve the acquisition of software intensive systems has been conducted by , but they focused on large scale governmental procurements. Since the largest number of the procurements is made by small and medium sized organizations, thus additional research is needed. To elicit these needs, this section identifies six steps involved in basic procurement of tool, which addresses problems and challenges experienced by small and medium sized organizations procuring software intensive systems.

A) The Procurement cycle

The procurement of software can be regarded as a never-ending cyclical process. During the monitoring and evaluation of operational systems, i.e. installed, accepted

and running systems, the detection of the need for new software appears.

This cycle can be segregated into six step procurement process where several approaches can be used to detail the process. These are as listed below

- Requirement - Defines a set of requirements for the intended software system. These requirements are meant to ensure that the final system will meet the needs detected in the previous phase by describing the required functionality.
- Evaluate responses – The evaluation is performed after collecting responses from the suppliers, to determine the ones that have met the mandatory requirements. Suppliers' failing to meet these requirements are eliminated.
- Contract negotiation - The supplier of the software product negotiates with the management in order to agree upon details of delivery. The contract should be legally enforceable agreement and include guarantees and criteria in accordance with the requirements defined in the initial phase.
- Installation/ Testing/ Acceptance - This phase is to give the procurer the possibility to test the system before final acceptance. Criteria for acceptance should be defined in the requirement phase and agreed upon in the contract. When the procurer is satisfied with the delivered system, the procurement project is signed-off.

B) Risk Management In Software Procurement

We have seen that "exposure to the consequences of uncertainty. It includes the possibility of loss or gain, or variation from a desired or planned outcome, as a consequence of the uncertainty associated with following a particular course of action". To be able to do this, risk management plans and procedures are helpful tools. This is an ongoing task throughout the whole project, and has to be monitored continuously. The following are approaches to project risk management according to;

i) Establish the risk context

The purpose of this step is to gather the information needed to establish a structure for the execution of the following steps in the process. The input to this activity is documentation describing the purpose and scope of the project . Identification and analysis of the stakeholders is included in the context establishment. This is done to get an overview of all parties involved in the project, and also to be able to evaluate their needs in relation with the requirements.

ii) Identify the risks

This process must be extensive and thorough, so that as many risks as possible are identified. Risks that are not recorded during this phase will not be assessed in the following phase, and they might threaten the success of the project later on. During the risk identification, the key elements defined in the first step of the process ease the systematical examination of the project .

iii) Analyzing the risks

When all the risks are identified, an assessment of the risks is performed. This assessment can be performed qualitative, semi-quantitative or quantitative. One of the activities that are conducted during the risk analysis phase is the establishment of a priority-setting matrix. This matrix is used in the priority rating of the risks later on. The priority-setting matrix can be of different sizes, depending on the desired scaling.

iv) Evaluating the risks

The purpose of the risk evaluation step is to establish a final documentation of the risks, including rating, treatment actions and the name of the responsible person. Some risks may involve inherent risks if they occur. Procurers often face fixed budgets in software procurement projects . With this limitation, the risk prioritizing becomes an important task.

v)Treating the Risks

The final step in the risk management process is to determine, which mitigating actions should be carried out to reduce the risk exposure.

IV. PROPOSED MODEL

The simulator will calculate the effort for the project over several simulation runs. After calculated efforts there will be clear idea to reuse managers how to allocate resources. And after calculated optimal size, we can measure the optimal efforts for the software. On the basis of efforts estimation reuse managers can decide how to allocate resources to the different activities, how to divide work within a phase. We can modify the parameter size. The concept of reuse plays very important role in the industries.

Effective size of existing software can be calculated as

Effective size = existing size * (0.4 * redesign% + 0.25 * reimplementation% + 0.35 * retest %)

Effective size of new software can be calculated as

Effective size1 = new code + existing size * (0.4 * redesign% + 0.25 * reimplementation% + 0.35 * retest %)

Reuse% = (RSI/Total Statement) * 100

RSI (Reused Source Instruction)

Formula used for calculation the efforts for the project is

EFFORT = a * Effective Size of a project measures in KLOC (Kilo Lines of Code).

EAF is effort adjustment factor, which will be calculated using cost drives. A cost estimation model calculates efforts using a function of program size and a set of cost drivers attributes.

Value of a, b depends on the complexity of software.

EAF which will be calculated using cost drivers factors.

The projects are categorized into three types:

a. Organic

b. Semidetached

c. Embedded

These categories roughly characterize the complexity of the project with organic projects being those that are relatively straight forward and developed by a small team, and embedded are those that are ambitious and high

requirements for such aspects as interfacing and reliability[6].

The Steps are to be followed from the start of Testing of software to the end of the testing as follows:

1) Before the dynamic testing, there is a static testing. Static testing includes review of documents required for the software development. This includes following activities. Static testing includes review of documents required for the software development. This includes following activities:

(a) All the documents related to customer requirements and business rules that are required for software design and development should be handed over to QA signed by the Project Manager.

(b) QA reviews these documents. The reviewing of documents includes comprehensive and thorough study of the documents

(c) After this there should be a formal meeting between the QA and development team regarding these documents, the agenda of this meeting mainly includes what is missing in the document, QA queries to be answered by Development/Project Team and/or clarification required for any confusions.

2) After the Software development or build of a module, QA starts dynamic testing. If during the development the requirement has been changed on customer demand or due to any other reason, then that should be documented and a copy of this revised document is given to the QA and also discussed as explained in point 1 above.

3) Development and Testing environment should be made clear to the QA by the Development team. It include the following activities:

(a)Server to hit for Testing

(b)Installation of latest build on the test server.

(c)Modules/Screens to test.

(d)Test duration as decided by test manager and project manager mutually based on scope of work and team strength.

(e)Demo of the software on test server by development team to the QC members.

4) After this Test cases and test scenarios are prepared and then the Test execution by QC.

5) A comprehensive Report of Bugs is prepared by the Testers and a review/verification by QC/QA/Testing Head takes place. Before handing over this report to Development Team there is a thorough review of Bugs List by Test Manager and in case of any clarification required on a bug submitted, the Testing Head discusses the bugs with the assigned tester.

6) Release of bug report by QC Team to Development Team.

7) Discussion/simulation of bugs by QC with development team if development team requires and time required for fixing the bugs should be made clear by Dev team at this stage.

8) Feedback from Development team on reported bugs with the stipulated time frame required to fix all bugs.

9) Any changes in the software being made in respect to fix these bugs should be made clear to the QA team by the Development team.

10) Testing team then Retests or verifies the bugs fixed by the development team.

11) Submitting the retesting bug report to the Test manager and after this the step 5 to step 10 are followed until the product has reached a stage where it can be released to customer.

12) Criteria for ending the testing should be defined by management or Test Manager Like when all major bugs are reported and fixed. Major bugs mean the bugs that affect the Business of the Client.

V. NEW SOFTWARE PROCESS MODELS

A. C.RAD Model

RAD is a linear sequential software development process model that emphasis an extremely short development cycle using a component based construction approach as shown in figure 1. If the requirements are well understood and defines, and the project scope is constraint, the RAD process enables a development team to create a fully functional system with in very short time period

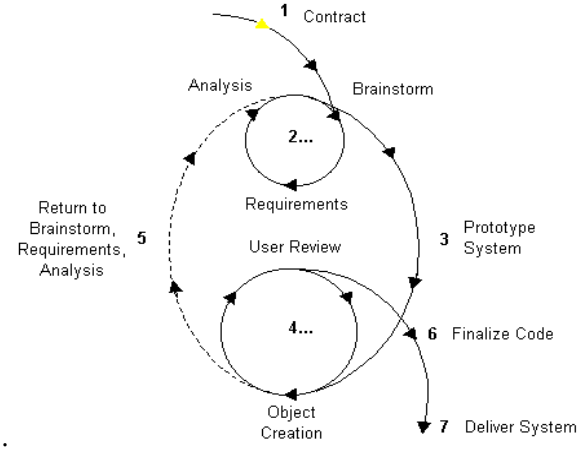

Figure 1. RAD development Model

This model assumes the requirements to remain static during the life of the project, so there is little or no chance of incorporating new changes to the software once work begins. If changes are tried to be incorporated it leads to more confusion and further delays.

The major weakness of the Waterfall Model as in figure is that it does not allow for much reflection or revision. Once an application is in the testing stage, it is very difficult to go back and change something that was not well-thought out in the concept stage. Often the product that is implemented at the end of the process is obsolete as it goes into production[1].

B . Incremental Model

The incremental model is a method of software development where the model is designed, implemented and tested incrementally until the product is finished. It involves both development and maintenance This model combines the elements of the waterfall model with the iterative philosophy of prototyping as shown in figure 2. That is basic

requirements are addressed, but many supplementary features (some known, others unknown) remain undelivered.

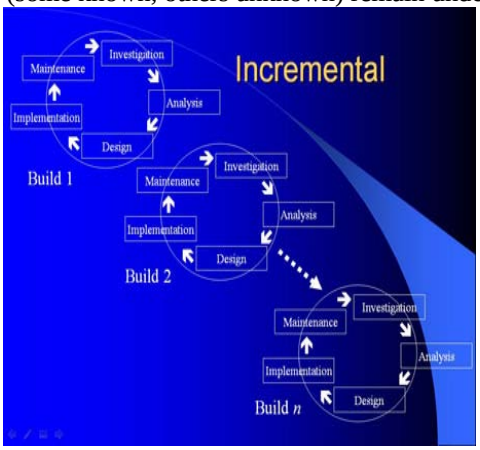

Figure 2.The Incremental model.

A lot of problem may become the reason of unsuccessful project. Lack of management, Lack of well Communication, Lack of well communication between costumer and organization, Technical problem and lack of resources, Lack of man power, not provided a good training.

C. Software Reuse Model

Software reuse is the process of creating software systems from existing software rather than building them from scratch Software reuse is still an emerging discipline. It appears in many different forms from horizontal reuse and vertical reuse to systematic reuse, and from white-box reuse to black-box reuse. Many different products for reuse range from ideas and algorithms to any documents that are created during the software life cycle Source code is most commonly reused in software systems; thus many people misunderstands software reuse as the reuse of source code alone. Recently source code and design reuse have become popular with (object-oriented) class libraries, application frameworks, and design patterns. Software components provide a vehicle for planned and systematic reuse.

Need to Reuse Software

A good software reuse process facilitates the increase of productivity, quality, and reliability, and the decrease of costs and implementation time. An initial investment is required to start a software reuse process, but that investment pays for itself in a few reuses. In short, the development of a reuse process and repository produces a base of knowledge that improves in quality after every reuse, minimizing the amount of development work required for future projects and ultimately reducing the risk of new projects that are based on repository knowledge. Software parts are shipped with the libraries available with SW. These SW parts are called components. Four levels of reuse are proposed:

1. code level components (modules, procedures, subroutines, libraries, etc.)
2. Entire applications
3. Analysis level products
4. Design level products

VI. THE TYPES OF SOFTWARE REUSE

Concerning motivation and driving factors, reuse can be:

- Opportunistic - While getting ready to begin a project, the team realizes that there are existing components that they can reuse.
- Planned - A team strategically designs components so that they'll be reusable in future projects.

Opportunistic reuse can be categorized further:

- Internal reuse - A team reuses its own components. This may be a business decision, since the team may want to control a component critical to the project.
- External reuse - A team may choose to license a third-party component. Licensing a third-party component typically costs the team 1 to 20 percent of what it would cost to develop internally. The team must also consider the time it takes to find, learn and integrate the component.

Concerning form or structure of reuse, code can be:

- Referenced - The client code contains a reference to reused code, and thus they have distinct life cycles and can have distinct versions.
- Forked - The client code contains a local or private copy of the reused code, and thus they share a single life cycle and a single version.

Fork-reuse is often discouraged because it's a form of code duplication, which requires that every bug is corrected in each copy, and enhancements made to reused code need to be manually merged in every copy or they become out-of-date. However, fork-reuse can have benefits such as isolation, flexibility to change the reused code, easier packaging, deployment and version management [4].

Systematic software reuse

Independent of what a component is, Systematic Software Reuse influences almost whole software engineering process. For providing guidance in the creation of high quality software systems at low-cost the software process models were developed. The original models were based on the (mis)conception that systems are built from scratch according to stable requirements. Software process models have been adapted since based on experience and several changes and improvements have been suggested since the classic waterfall model. With increasing reuse of software, new models for software engineering are emerging. New models are based on systematic reuse of well-defined components that have been developed in various projects. These trends are particularly evident in markets, such as electronic commerce and data networking, where reducing development cycle time is crucial to business success.

Horizontal reuse

Horizontal reuse refers to software components used across a wide variety of applications. In terms of code assets, this includes the typically envisioned library of components, such as a linked list class, string manipulation routines, or graphical user interface (GUI) functions. Horizontal reuse

can also refer to the use of a commercial off-the-shelf (COTS) or third-party application within a larger system, such as an email package or a word processing program.

Vertical reuse

Vertical reuse, significantly untapped by the software community at large, but potentially very useful, has far reaching implications for current and future software development efforts. The basic idea is the reuse of system functional areas, or domains that can be used by a family of systems with similar functionality. The study and application of this idea has spawned another engineering discipline, called domain engineering. Domain engineering is "a comprehensive, iterative, life-cycle process that an organization uses to pursue strategic business objectives. The form and structure of the application engineering activity are crafted by domain engineering so that each project working in a business area can leverage common knowledge and assets to deliver a high-quality product, tailored to the needs of its customer, with reduced cost and risk". Domain engineering focuses on the creation and maintenance of reuse repositories of functional areas, while application engineering makes use of those repositories to implement new products[9].

VII. RISK ANALYSIS AND MANAGEMENT AND CHALLENGES

Realistically, answers to the above questions will not be found in the research laboratory. More likely, they are to be found with the intelligent project manager who "knows the risks, their degree, their causes, and the action necessary to counter them, and shares this knowledge with colleagues and clients" (Gilb 1988).

The figure 3 illustrate detail classification on risk analysis. Indeed, a fundamental appreciation of the risks involved in the institutionalization of reuse technology, their analysis and management, can go a long way to ensuring successful reuse-based projects. As such, a good reference point is to familiarize ourselves with the basic risk analysis and management process, shown in Figure 1 (adapted from (Charette 1989)).

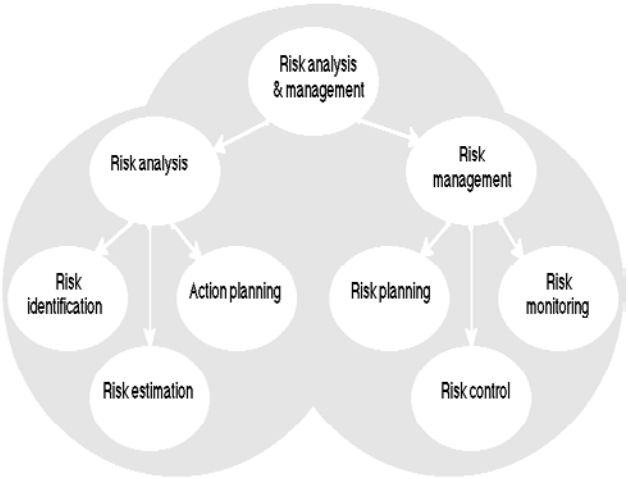

Figure 3: Risk Management Phases.

Risk analysis involves three tasks:

- Risk identification: Identifying potential problems before they occur.
- Risk estimation: Quantifying these problems, often in terms of some measure of severity.
- Action planning: Generating alternative choices of actions that would help prevent a problem occurring in the first place or reduce the adverse effects of the problem if it did occur.

Challenges

The basic premise of software reuse is support for design methodologies for which the main activity is not the building of new systems from scratch, but the integration, modification, and explanation of existing ones [Winograd, 1996]. Software reuse is a promising design methodology because complex systems develop faster if they can be built on stable subsystem [8].

Benefits of Software Reuse

-Increased dependability
-Reduced process risk
- cost of development
- Effective use of specialists
-Standards compliance

CONCLUSION

Major issues in component and software testability are modeling component testability, providing support for user oriented testing. Modeling component testability rests on providing operational definition of intuitive factors. This helps in devising better methods and comparison of two or more techniques proposed for same goal. However, collaboration with other companies is another major factor affecting the selection process of tool. This is an immense factor as it brings the experience of other partner company with tool and accountable for providing competitive advantage. The central thesis of this paper is that contemporary models of software development must account for software the interrelationships between software products and production processes, as well as for the roles played by tools, people and their workplaces. New models for software development enabled by the Internet, group facilitation and distant coordination within open source software communities, and shifting business imperatives in response to these conditions are giving rise to a new generation of software processes and process models The development of complex software systems is challenging design activity. The process is made difficult not because of the complexity of technical problems, but because of the social interaction when users and system developers learn to create, develop and express their ideas and visions.

REFERENCES

[1] Stefan Kuhlmann and Uwe Kuntze (1990) ”R&D-cooperation by small and medium sized companies” Fraunhofer-Institut für Systemtechnik und Innovationsforschung (FhG-ISI) Technology Management : the New International Language. Page(s): 709-712.

[2] Knut Steinar Engene (2005) “Towards improving an organization's ability to procure software intensive systems”. Master thesis at the Department of Computer and Information Science at The Norwegian University of Science and Technology. 28. june 2005. Norway.

[3] IEEE Recommended Practice for Software Acquisition. IEEE Std 1062, 1998 Edition. The Institute of Electrical and Electronics Engineers, Inc. ISBN 0-7381-1514-2. [1] R. S. Freedman, “Testability of Software Components,”IEEE Trans. Softw. Eng., vol. 17, June 1991, pp. 553- 564, doi=10.1109/32.87281.

[4] E. J. Weyuker, “Testing Component-Based Software: A Cautionary Tale,” IEEE Softw., vol. 15, September1998, pp. 54-59, doi=10.1109/52.714817

[5] Cagatay Catal., “Software fault prediction: A literature review and current trends”, Expert systems with applications Volume 38, Issue 4, April 2011, Pages 4626-4636 Elsevier.

[6] Evett.M., Khoshgoftar.T, Chien.P and Edward Allen, “GP-based software quality prediction”, in 3rd annual conference on genetic programming PP. 60–65, 1998.

[7] Khoshgoftaar. T, Gao.K and Szabo.R.M, “An application of zero-inflated poisson regression for software fault prediction”, In 12th international symposium on software reliability engineering, Washington, DC: IEEE Computer Society, PP 66–73, 2001.

[8] Martin L. Griss, John Favaro, and Paul Walton. Managerial and Organizational Issues - Starting and Running a Software Reuse Program, chapter 3, pages 51-78. Ellis Horwood, Chichester, GB, 1993.

[9] Martin L. Griss. Software reuse: from library to factory. IBM Systems Journal, 32(4):1-23, November 1993.

[10] Martin L. Griss and Kevin D. Wentzel. Hybrid Domain-Specific Kits for a Flexible Software Factory. In proceedings: SAC'94, Phoenix, Arizona, March 1994.

[11] J. E. Gaffney and R.D. Cruickshank. A General Economics Model of Software Reuse. In proceedings: 14th ICSE, Melbourne Australia, May 1992.

#1 P. Ranjeet Kumar is Pursing B.Tech Second year in Computer Science Engineering from KKR & KSR Institute of Technology and Sciences [KITS], Vinjanampadu, Guntur., A.P, INDIA, and Affiliated to Jawaharlal Nehru Technological University (JNTU) , Kakinada, A.P, India.

#2.Professor R.Ramesh received B.E in Computer Technology and Engineering from RVR&JC, Nagarjuna university, India, holds a M.Tech in Computer Science from Jawaharlal Nehru University New Delhi. He has 12 years of vast experience in Computer Science and Engineering areas.

#3.Professor T.Venkat Narayana Rao, received B.E in Computer Technology and Engineering from Nagpur University, Nagpur, India, M.B.A (Systems), holds a M.Tech in Computer Science from Jawaharlal Nehru Technological University, Hyderabad, A.P., India and a Research Scholar in JNTU. He has 21 years of vast experience in Computer Science and Engineering areas pertaining to academics and industry related I.T issues. He is presently working as Professor, Department of Computer Science and Engineering, Guru Nanak Institutions Technical Campus, Ibrahimpatnam, R.R.Dist., A.P, INDIA. He is nominated as an Editor and Reviewer to 28 International journals relating to Computer Science and Information Technology and has published 39 papers in international journals. He is currently working on research areas, which include Digital Image Processing, Digital Watermarking, Data Mining, Network Security and other emerging areas of Information Technology. He can be reached at tvnrbobby@yahoo.com

#4 Shireesha Dara , B.Tech and M.Tech. in comuter science and Enginering , ssistant Professor, Department of C.S.E, Guru Nanak institutions Technical Campus.Ibrahimpatnam. RR Dst. AP. She has 11 Years of Experience in academia and industry. She has been actively involved in coordinating and organizing plenty of national events such as seminars and workshops and published two papers. She has contributed in lab manuals and software programming at high-end computer centers. Worked as Organizing Committee Member for NBA tasks. She has also presented papers during the conferences to her credit and a meticulous guide to UG and P.G engineering projects. She can be reached at: sirisha.dara@gmail.com