Software Quality Modeling and Current State of the Art

N. Rajasekhar Reddy, R.J.Ramasree

Abstract: ***Software Quality Assurance plays a key role in software development. The research is mainly aimed at considering prior researches, present working status and to restore the gaps between them with present known information. Here, we conduct a review on current state of the art in software quality evaluation and assurance models.***

Keywords: SQA, Product metrics, software science, size-defect relationship, measurement applied to SQA, Terms—Software as a service (SaaS), software selection, service utility

I. INTRODUCTION:

Research on software quality is as old as software research itself. As in other engineering and science disciplines, one approach to understand and control an issue is the use of models. Therefore, quality models have become a well-accepted means to describe and manage software quality. Beginning with hierarchical models proposed by Boehm et al. [1], over the last 30 years, a variety of quality models has been developed, some of which have been standardized. Many of these models are used, for example to aid the specification of quality requirements, to assess existing systems or to predict the defect density of a system in the field. The last three decades in quality modeling generated a multitude of very diverse models commonly termed "quality models". Examples on the spectrum of diverse models include taxonomic models like the ISO 9126 [2], metric-based models like the maintainability index (MI) [3] and stochastic models like reliability growth models (RGMs) [4]. On first sight, such models appear to have little relation to each other although all of them deal with software quality. We claim that this difference is caused by the different purposes the models pursue: The ISO 9126 is mainly used to define quality, metric-based approaches are used to assess the quality of a given system and reliability growth models are used to predict quality.

To avoid comparing apples with oranges, we propose to use these different purposes, namely definition, assessment and prediction of quality, to classify quality models. Consequently, we term the ISO 9126 as definition model, metric-based approaches as assessment models and RGMs as prediction models. Although definition, assessment and prediction of quality are different purposes, they are obviously not independent of each other:

Manuscript Received January 09, 2012.

N. Rajasekhar Reddy, Department of Computer Science and Engineering, Madanapalli Institute of Technology and Science, AndhraPradesh, India (E-mail: rajsekhar007@gmail.com)

R.J.Ramasree, Department of Computer Science, Raastriya Sanskrit dyapeeta,Tirupati, AndhraPradesh, India (Email: rjramasree@yahoo.com)

It is hard to assess quality without knowing what it actually constitutes and equally hard to predict quality without knowing how to assess it. This relation between quality models is illustrated by the DAP classification shown in Fig. 1.

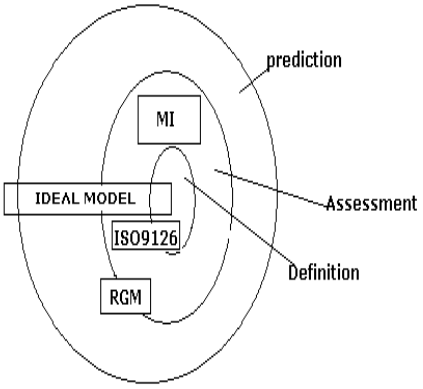

Fig1: DAP classification

The DAP classification views prediction models as the most advanced form of quality models as they can also be used for the definition of quality and for its assessment. However, this view only applies for ideal models. As Fig. 1 shows, existing quality models do not necessarily cover all aspects equally well. The ISO 9126, for example, defines quality but gives no hints for assessing it; the MI defines an assessment whose relation to a definition of quality is unclear. Similarly, RGMs perform predictions based on data that is not explicitly linked to a definition of quality.

In this paper we explore the Software Quality models and current state of the art.

II.SOFTWARE QUALITY MODELS AND CURRENT STATE OF THE ART

One of the main shortcomings of existing quality models is that they do not conform to an explicit meta model. Hence the semantic of the model elements is not precisely defined and the interpretation is left to the reader.

Quality models should act as a central repository of information regarding quality and therefore the different tasks of quality engineering should rely on the same quality model. But today, quality models are not integrated into the various tasks connected to quality.

For example, the specification of quality requirements and the assessment of software quality are usually not based on the same models. Another problem is that today quality models do not address different views on quality. In the field of software engineering, the value-based view is typically considered of high importance [5]. This view is largely missing in current quality models [6].

The variety in software systems is extremely large, ranging from huge business information systems to tiny embedded controllers. These differences must be accounted for in quality models by defined means of customization. In current quality models, this is not considered [7, 8, and 9].

III. DEFINITION QUALITY MODELS

Existing quality models lack clearly defined decomposition criteria that determine how complex concepts of quality are to be decomposed. Most definition models depend on a taxonomic, hierarchical decomposition of quality attributes. This decomposition does not follow defined guidelines and can be arbitrary [10, 11, 12, 6]. Hence, it is difficult to further refine commonly known quality attributes, such as availability. Furthermore, in large quality models, unclear decomposition makes locating elements difficult, since developers might have to search large parts of the model to assert that an element is not already contained. This can lead to redundancy due to multiple additions of the same or similar elements.

The ambiguous decomposition in many quality models is also the cause of overlaps between different quality attributes. Furthermore these overlaps are often not explicitly considered. For example, security is strongly influenced by availability (denial of service attack) which is also a part of reliability; code quality is an important factor for maintainability but is also seen as an indicator for security [13].

Most quality model frameworks do not provide ways for using the quality models for constructive quality assurance. For example, it is left unclear how the quality models should be communicated to project participants. A common method of communicating such information is guidelines. In practice, guidelines that are meant to communicate the knowledge of a quality model experience various problems. Often these problems are directly related to corresponding problems of the quality models itself; e.g. the guidelines are often not sufficiently concrete and detailed or the document structure of the guideline is not aligned according to an evident schema. Also rationales are often not given for the rules the guidelines impose. Another problem is that the quality models do not define tailoring methods to adapt the guidelines to the application area.

IV. ASSESSMENT QUALITY MODELS

The already mentioned unclear decomposition of quality attributes is in particular a problem for analytical quality assurance. The given quality attributes are mostly too abstract to be straightforwardly checkable in a concrete software system [3,5]. Because the existing quality models neither define checkable attributes nor refinement methods to get checkable attributes, they are hard to use in measurement [14, 6].

In the field of software quality, a great number of metrics for measurement have been proposed. But these metrics face problems that also arise from the lack of structure in quality models. One problem is that despite defining metrics, the quality models fail to give a detailed account of the impact that specific metrics have on software quality [6]. Due to the lack of a clear semantics, the aggregation of metric values along the hierarchical levels is problematic. Another problem is that the provided metrics have no clear motivation and validation. Moreover, many existing approaches do no respect the most fundamental rules of measurement theory and, hence, are prone to generate dubious results [15].

Due to the problems in constructive and analytical quality assurance, also the possibility of certification on basis of quality models experiences elementary problems [14]. It has to be noted that measurement is vital for any control process. Therefore the measurement of the most important quality attributes is essential for an effective quality assurance processes and for a successful requirements engineering.

V. PREDICTION QUALITY MODELS

Predictive quality models often lack an underlying definition of the concepts they are based on. Most of them rely on regression using a set of software metrics. This regression then results in equations that are hard to interpret [16]. Furthermore, prediction models tend to be strongly context-dependent, also complicating their broad application in practice. Many factors influence the common prediction goals and especially which factors are the most important ones varies strongly. Usually these context conditions are not made explicit in prediction models.

VI. DISTRIBUTED SYSTEM QUALITIES EVALUATION APPROACH

As service-oriented distributed systems grow in size and complexity, ensuring that they conform to their specifications throughout the software life cycle becomes more difficult. This difficulty stems in part from the problem of serialized- phasing development [17], in which application- level entities are developed after infrastructure-level entities. Serialized-phasing development makes it difficult to evaluate end-to-end functional and quality-of-service (QoS) aspects until late in the software life cycle for example, at system integration time.

Agile techniques help address functional aspects of serialized-phasing development by validating software functionality throughout the entire software life cycle[18, 19]. For example, test-driven development and continuous integration are agile techniques that validate functional quality by ensuring that software behaves as expected. The benefit of using agile techniques to improve QoS assurance of service-oriented distributed systems, however, has not been demonstrated. Developers, therefore, need new techniques to help alleviate the complexity of serialized-phasing development and enable evaluation of QoS concerns throughout the software life cycle.

Model-driven engineering (MDE) is a promising solution for improving software development of

service-oriented distributed systems [20]. MDE techniques, such as domain-specific modeling languages (DSMLs)[21], provide developers with visual representations of abstractions that capture key domain semantics and constraints. DSMLs also provide tools that transform models into concrete artifacts, such as source code or configuration files that are tedious and error-prone to create manually using third-generation languages. Moreover, such artifacts often are not available early enough in the software life cycle to allow a proper evaluation of end-to-end QoS properties.

James H. Hill et al[22] discuss about the way in which Domain Specific Modeling Languages (DSMLs) are used to realize agile techniques for continuously evaluating service-oriented distributed-system Quality of service (QoS) throughout the software life cycle. As service-oriented distributed systems grow in size and complexity ensuring that they confirm to their specifications throughout the software life cycle becomes more difficult which stems in part from the problem of serialized-phasing development which application level entities are developed after infrastructure-level entities. Evaluation of end-to-end functional and QoS aspects are difficult due to serialized-phasing development until late in the software life cycle.

Address functional aspects of serialized-phasing development are helped by Agile techniques. They validate software functionality throughout the entire software life cycle. A promising solution for improving software development of service oriented distributed systems is Model-driven engineering (MDE). DSMLs which is a MDE technique provides developers with visual representations of abstraction that capture key domain semantics and constraints. Agile QoS assurance process is by system execution modeling in which the developers rapidly model the behavior and workload of the distributed systems being developed, synthesize a customized test system from models, execute the synthesized test system on a representative target environment test bed to produce realistic empirical results at scale and analyze the test system's QoS in the context of domain specific constraints.

Component workload emulator [coworker] test suite (CUTS) is a operating system, programming language, middleware-independent DSML-based system execution modeling tool for service oriented distributed systems. QoS can be used in service oriented distributed systems by using Component Behavior Modeling Language (CBML) DSML models component behavior, the Workload Modeling Language (WML) DSML models component workload and the Understanding Nonfunctional Intensions Via Testing Experimentation (Unite) DSML species QoS unit tests for performance analysis in distributed systems.

The QoS-Enabled Dissemination (QED) project applies these CUTS DSMLs and their agile techniques. Under the first capability capturing behavior and workload is studied, under the second capability the realistic data is generated and the third capability focuses on collecting and analyzing distributed system data.

The service-oriented distributed systems respond to inputs like events or remote method invocations and are reactive. This system's workload and behavior are analogous to sequence of actions that cause side-effects DSML based system execution modeling tools must use intuitive domain-specific abstractions and they must capture properties to provide a light weight adaptive process. The behavior and workload of the received target event begin with an input action followed by a sequence of actions and states defines the behavior. QED developers can easily adapt their models to test different scenarios as they model behaviors and workload using DSMLs. DSMLs facilitate in saving time decreasing errors.

Ensuring QoS of service-oriented distributed system require continuously generating realistic data and results. DSML based system execution modeling tools must provide these information leveraging model interpreters made the realization of capability in CUTS. After QED developers use CBML and WML to model the behavior and workload of components in the multistage work flow application, they use CUTS model interpreters to generate source code customized to their target environment.

Collecting and analyzing QoS metrics in service-oriented distributed systems is difficult because the data often changes overtime. Systems structure can also change over the life cycle. Data collection and analysis technique must adapt to the volatility in service oriented distributed systems. Log formats, casual relations, user-defined evaluation function are used to define QoS unit tests.

Aggregation function like SUM(f) and AVG(f) combines multiple occurrences of a result and a grouping criteria, which partition results into sets before aggregation. QoS unit test's data trend can be viewed throughout the system's execution in its target environment, by removing the aggregation function. Average service time on the basis of the partial specification is calculated with the help of an equation which is f = AVG(LF2.sendTime, LF1.recvTime). QED developers are provided with a lightweight technique to ensure QoS by automatically extracting metrics of interest from system traces by the Unite DSML. When analyzing extracted data Unite doesn't require the QED developers to understand distributed-system composition.

Experiments were conducted to evaluate the QoS of the QED and Global Information Grid (GIG) middleware by applying the CUTS DSMLs to a multistage workflow application consisting of six different component stages that runs atop GIG middleware. Each application component contains a CUTS behavior and workload model that stresses different parts of the QED and GIG middleware. Each component also contains actions that log metrics for Unite to analyze the QED and GIG middleware performance. Experiments were conducted in a representative target environment test bed at the Institute for Software Integrated Systems (ISIS) ISISlab.

Observation: The existing QoS capabilities of the GIG middleware are determined with the help of responsive time which also gives information about where the QED middleware can improve QoS relative to the GIG middleware baseline. Application of agile techniques in DSML-based system execution tools facilitates in reduction of the effort of testing service-oriented distributed-system QoS. Less time and

effort are required by the QED developers in order to generate and run tests in their target environment than to implement it manually. The complexity of analyzing results for the distributed systems using Unite is alleviated by CUTS agile techniques. The number of log formats for the QoS unit tests remains constant as the system complexity grows. Unite's QoS unit test specification process is a one-time effort for software developers when the log formats are assumed to remain stable.

VII.SOFTWARE MODULES QUALITY ASSURANCE APPROACH BASED ON SIZE AND DEFECT RELATION

As the truth prevails that software is in the phase of tremendous development and has been making us highly dependable, it's necessary to look into the defects and minimize them and one such defect as per the research is that higher the size of software, higher the defects. Hence, with the help of special oriented products like Mozilla, Cn3d, jboss and eclipse we investigate functional form of the size. Understanding the importance of the relationship between the size and defect proneness of software modules, and how its relationship would facilitate various development decisions linked to prioritization of quality assurance activities.

Gunes koru et al[23] conducted study about various relationships that is between the size and defects linearly, size and defects non-linearly, size and defects in terms of density and finally, size and defects without any relationship and to study these we adopt various methods, mainly about implementation of conventional data having various factors like deleted modules which are problematic as their defect counts are smaller than the non deleted ones and They might also be removed from the data analysis but doing so might decrease the number of data points, and size changes which study about Ignoring size changes. For example, a module measured when it was small can become larger and more defect prone over time. Then a COX is introduced to avoid the potential internal validity threats. But, in this study a conditional COX is used.

Then the size and fault of data is used in the study followed by a data format description.

The initial product, Mozilla, is a Web browser and was a large-scale product, and Cn3D, which is a smaller product and Cn3d is a bioinformatics application for Web browsers written in C++ which visualizes 3D structures from the National Center for Biotechnology. Unlike Mozilla, which is a general-purpose product, Cn3d is a domain-specific product and eclipse is a Java Integrated Development Environment (IDE) that has a large suite of sub-products and finally, JBoss is one of the most commonly used Java application servers. JBoss data set included 9,428 observations that belonged to 1,703 Java classes.

The data format description, with this process either ends the period or a class is deleted in this and after this modeling and outputs are studied wherein we study various COX models and the tests applied to it and study the importance of modeling output to size-defect relation. the findings have many important uses. They show that, when working on the focused quality assurance works like testing and code inspections, it will be useful to give higher priority to smaller modules. Hence, suggestion is that practitioners choose among practitioners choose among different prioritization strategies by also considering that smaller modules are proportionally more defect prone.

Keeping the various threats to validity in mind, like (i) construct validity in which the defect proneness was operated with the risk of a defect fix for classes. We identified defect fixes from the CVS logs entered by developers, some issues may arise, like some defects may not surface, some defects may surface but not get fixed, and some defect fixes may not be recorded in CVS logs, (ii) internal validity in which Some other variables may influence on defect proneness. Hence, the internal validity threat caused by not using other structural measures is to be minor. Anyhow developers' skills, expertise, and training will definitely affect the defects. Hence, Cox modeling can't be avoided here when some factors are unknown or uncontrollable. (iii) External validity, in which we validate our findings from mozilla by using the size and defect data collected from three other open-source products. The replicated studies examining this relationship on other software products will for sure be useful to assess our findings after gaining confidence in the external validity of the results reported. (iv)Finally narrowing down to the conclusion validity wherein, even though there are potential threats to the validity of this study just like any other empirical study, we have confidence that our study and findings have adequate validity.

Observation: Hence from the above discussion it is to be understood that the functional form of the relationship between software size and defect is vital for development and inspection with limited amount of resources for quality assurance. It is also clear that smaller modules are proportionally problematic compared to the larger ones. This study is also useful to software practitioners when they decide about modules to be chosen for focused testing and inspection. The practitioners should consider giving high priority to smaller modules to increase the effectiveness of their quality and to use the resources efficiently. The replicated studies will be useful to assess the findings and to test if the phenomenon can be stated as a theory or not. Earlier, it has been found out that the interface defects that were connected with structures present outside the modules local environment, could explain why smaller modules are proportionally more troublesome because, in the system they studied, the interface defects were equally distributed over smaller and larger modules which is important for further investigation because, if validated, it gives an explanation about the mechanisms behind the size-defect relationship discovered here.

VIII.SOFTWARE SERVICE QUALITY ASSESSMENT APPROACHES

Service selection is a multi criteria decision-making problem whose resolution commonly involves a trade-off between quality and cost. As explained before, there is no guarantee of service quality at selection time; however, reputation can help in predicting the likelihood of a quality offer to be met. As a matter of fact, selection can translate into a three-

criterion decision-making problem involving reputation, quality, and cost. This problem can be reduced to a single-criterion decision-making problem provided that quality reputation and cost are aggregated into a single selection metric. Although recent literature works agree on the necessity of considering reputation for service selection, there is no general consensus on the role of reputation in decision support. Considering service reputation as the aggregation of "arbitrary" consumers' feedback makes it hard to clearly define what exactly feedback refer to (credibility, reliability, etc.) and what exactly reputation stands for.

Ensuring the veracity of reputation reports is also a critical issue. First, feedback can be subjective since it is based on consumers' "personal" expectations and opinions. Second, consumers may have an "obstructed" view of a service and its performance, especially when the latter is part of a composite service. Third, reputation systems are prone to attacks by malicious consumers who may give false ratings and subvert service reputation. Generally, it is harder to maintain a per-consumer reputation system than a per-service reputation system, mainly because services are less versatile, more traceable, and come in a smaller number. Moreover, it is harder to manage user identities especially for malicious users who are likely to change their quite often (e.g., sybil attacks [6]). Finally, consumers may have little incentive to leave feedback; they are often more eager to leave negative feedback when they are dissatisfied with the experienced service than to leave positive feedback when they are satisfied. This introduces a bias against positive ratings and leads to unfair reputation reports. For all of these reasons, the first step toward establishing the foundations of an automated reputation aware selection framework is to unambiguously define the feedback as a computable no arbitrary metric and to devise an objective rating system. As continuity to models explored above we review the work carried out by Limam, N et al [26].

The widespread use of internet, the software as a service (SAAS) model in which software is delivered on demand and priced on use has become widespread over the market. As discussed in the paper the evolution of SaaS model is the result of outsourcing software in the development of project which is challenging as well as risky, the risk factor is because of the performance or quality of the external software which at some point of application may not deliver the expected quality. Hence, for testing the quality of software SaaS is a model which provides a low-risk alternative of COTS models in large investments.

The motive of the paper framework lies in the fact that service selection is a multi-criteria decision making problems and there is no guarantee of service quality at selection time. Hence, a service selection is based on three parameters: - reputation, quality and cost.

The work presented in this paper discusses risk management pursuant to project development through extended service software components. The paper presents an automated quality and reputation based framework, which is independent of consumer ratings for service rating and selection in Software as a Service (SaaS). The work of this paper is different from the previous work from the fact that it takes into consideration of automation of the service rating process. The service proposed in this paper facilitates the user to take feedback which has been assigned to a delivery service that objectively reflects the satisfaction or dissatisfaction with the offered service and quality.

The degree of coherence between the service quality provided and promised is monitored and translated into a utility metric. For the fulfillment of this objective, a feedback computation model is derived from the expectancy-disconfirmation theory from market science, this model further generate feedback based on the service utility and cost. Moreover, for better and more accurate presentation of feedback results a model named as, reputation derivation is proposed which integrates feedback with the reputation value which reflects the performance of the service at selection time. The most important characteristic of work is the service ranking function that it integrates the quality, cost and reputation parameters into a single metric that is used to evaluate service offering against each other, so that the ranking has to be evaluated against only one parameter i.e. the integrated parameter proposed in this paper.

Observation: The objective of adding the service selection process is accomplished by the authors through a rating function which provides feedback on delivered service without human intervention. The mathematical formulation involved in the quality monitoring functions and the various variables are as follows:-

ν is utility function and is defined as a weighted product of the utilities associated with each parameter Qi, ν is expressed as-

$$\nu = \prod_{Q_i \in Q_0 S_{\dim}} F_{Q_i}^{cQ_i}$$

Where, FQi is a function that gives the utilities associated with the parameter Qi.

Qi : quality parameter

CQi ϵ [0,1] reflects how much the users cares about quality parameter.

$$Accept_i = \left\{ \begin{matrix} dom(Q_i) \to 0,1 \\ q_i \to Accept_i(q_i) = \begin{cases} 1 \text{ if } q_i \text{ better than or wqual to } (q_i)_e \\ 0, \text{ otherwise} \end{cases} \end{matrix} \right\}$$

For utility computation a function parameter 'Accepti' gives results based on the domain of the quality parameter Qi and the agreement index, qi. The agreement index is the equivalence in the promised and the provided quality. The function 'Accepti' follows a Bernoulli distribution pi to meet the quality qi represented by the quality parameters.

IX.SQ EVOLUTION MONITORING APPROACHES FOR DEFECTS

Hongyu Zhang [27] proposed Chart based Software quality evolution model for defects. This can be considered as current state of the art in chart based models. The review of this work fallows.

Unlike all other works, which mostly deal with only development process and its functionality, this article is dealing with Quality of software to be evaluated. This includes the calculation of reusability, maintainability of software that is developed. These times the research interests and work is more on those areas which include software evolution.

The adoption of SPC(software process control) is well explained using two very good and well formed open system software namely ECLIPSE and GNOME. The technique here followed is the use of different control charts. This technique is good and is apt for calculating the quality. Here, evaluation is done in accordance to graph between the number of bugs or defects over a certain period of time. C-charts usage is given much importance here in the article.

Next comes in to discussion the evolution of Eclipse. The evolution is observed in accordance to their changes in months, days and year's .Other important factor to fix values in C chart is number of source lines that change over that period. The use of Bugzilla to report and track defects is perfect. Over all the evolution process over years eclipse versions 2.1, 3.0, 3.1 have evaluated with increased quality and less bugs.

The evolution process of Gnome, which also follows the similar trends described above. But here the stability is less and in time of changing there is also a change in architectural design. Many versions like 2.0, v2.0, 2.0.0,(v19.0 to v19.8),(v2.0.1 to v2.0.5) have evaluated.

After the evolution of these patterns, the main focus is discussion of the pattern the charts have in plotting the defects. It has 6 kinds of patters namely downward trend, upward trend, Impulse, Hills, Small variations, Rollercoaster. All these are different in the number of bugs marked in C charts in time. As to consider, downward trend is decrease in number of bugs and increase in software quality. Upward trend is increase in number of bugs and decrease in quality. Impulse is sudden rise in bugs over a small interval of time.

Observation: Overall the article using C-charts, the quality is perfectly calculated .But it has no guaranty that it works successfully in other closed systems which have less variations compared to open systems ,Eclipse and Gnome. It also has not discussed anything about other charts like r-charts, x-chart and p-charts.

X.SOFTWARE MAINTENANCE QUALITY AND MINING APPROACH

Alexander Tarvo[28] proposed an interesting model that can be considered as current state of the art in this category. In this model presented by Alexander Tarvo, a detailed study is done on Binary Change tracer (BCT) which is a multipurpose tool that can mine vital information on a software system. Software maintenance activities result in system modifications that are distributed to customers as updates. Incorrect changes result in software regressions which are painful as they cause failures in already stable features and parts of the system. The best method to avoid regressions' negative consequences is extensive testing of all fixes. A tool or method that can predict the amount of risk for each fix is needed. This need is met by BCT which extracts information on all changes that have happened to Microsoft Windows. Each fix's risk of regression is predicted by a statistical model built with the help of data mined with BCT. It exploits features of a standard engineering process so it can be used for a variety of software projects.

Though research exists on risk prediction in software projects, researchers have concentrated on fault proneness prediction (FPP) models. They tend to predict which parts of the newly developed software system will be more prone to failures. According to FPP models a system consists of components described by metrics (numeric properties) for example, statistical models like decision tree or logistics regression. FPP models aren't designed to predict the risk for a particular software update. Prediction of software regressions is much less explored area due to difficulties of mining data on fixes. A statistical model requires detailed information on hundreds of prior fixes which is scattered through multiple data sources. To build a regression prediction model we need to develop a special program called a mining tool that performs these tasks. BCT is a specialized tool not only for extracting fix metrics but can be used for other purposes.

A bug record is created in the Bug tracking Database (BTD) when any change in Windows is necessary and each bug record has a unique identifier (bud ID) which contains fields describing change. After initial analysis of bug, the current version of the source code from the Version Control Systems (VCS) is downloaded and changes are made. The change passes some basic functionality testing and then the changed source files are integrated back into the VCS in the form of an atomic transaction or check-in. After fix development is complete the test engineer must test the change. The amount of testing is influenced by many factors but the most obvious is regression risk. If testing reveals any problems with the change, the test engineer asks the developer to fix them or else the test engineer marks the bug record closed and a software update can be built. BCT uses a decomposition of system into binary modules. BCT collects information on all changes in the system that occurred during the specified time interval. Analysis of a binary occurs in four steps:

1. Dividing the binary into components.
2. Extracting the history of code changes.
3. Associating code changes with bugs.
4. Storing the extracted data.

The results are accumulated until a complete history of changes is retrieved. A software system generates symbol files in addition to binaries and these files determine which chunk of a binary corresponds to a particular line in the source file. This information is used by BCT to retrieve a complete list of names of the source files used to build old and new versions of the binary. BCT restores componentization of the software system and it stores each component in one of the internal lists.

BCT retrieves all the versions newer than that source files oldest version but older than or the same age as source files newest version and performs a pairwise

comparison of these versions in ascending order. The changes are added to the dictionary of atomic changes that happened to binary. The history of source file changes is retrieved automatically. Code changes in the VCS are related to corresponding bug records in the Bug Tracking Databases (BTD) to retrieve information about why the changes are made. Analysis of BTD is more difficult for the tool. BCT stores all mined entities, their metrics and their links in the SQL database but it itself doesn't mine all code metrics for changed functions and binaries. MaX framework which is another tool designed by Microsoft to extract metrics.

Fix regression prediction (FRP) model which is based on logistic regression is developed to predict a fix's regression risk. The fix metrics are the metrics of corresponding bug record and all its related entities like number of changes the fix caused, experience of the developer who made the fix, general fix metrics. The model comprises of inherent coefficient values and contains hidden knowledge regarding which fix metrics are the best indicators of the possible regression. Classifiers make a few mistakes unlike the ideal classifiers which detect all high risk fixes without making mistakes. Each fix prediction has four possible outcomes which are true positive, false positive, true negative and false negative. These form the basis for several metrics for classifier accuracy. True Positive Rate (TPR) and False Positive Rate (FPR) are the two widely used metrics and have values between 0 and 1. A Receiver Operating Characteristic (ROC) curve is a graph with the TPR values on y-axis and FPR values on x-axis. These are used to estimate classifier performance and the area under ROC curve (AUC) is a number characterizing the classifier performance.

Observation: BCT reported metrics and determined whether the fix caused any regressions. The model's accuracy is measured by using a data splitting technique. A stepwise procedure is used to determine which metrics predict risk. BCT automatically extracts its metrics, passes them to FRP model to calculate the regression risk when a new fix is available. Fix's are categorized into very high risk, high risk, average risk, low risk and very low risk based on the risk. Test engineers use this information to plan fix testing. FPR model has significantly higher accuracy than test engineers; hence it is useful in in risk prediction. FRP model's utility is estimated by comparing its results with those from manual estimation of regression risk. The test engineer analysed the fix and classified its risk as high, medium or low. Build reports are being integrated with the FRP model and the model is used to predict regression risk for each bug fixed in a build which will help leverage testing of the daily builds. BCT is also used to analyze daily builds which provide a complete list of changes in the software project which are used to track its progress. BCT user interface will let us visualize all processes occurring in the software system.

XI. MULTIPLE REPOSITORIES CENTRIC SOFTWARE QUALITY MODELING APPROACH

Software quality models generally predict, for a program module, either the number of defects it is likely to have or the quality-based risk category it belongs to, e.g., faultprone (fp) or not-fault-prone (nfp) [29], [30], [31]. In the literature, various classification techniques have been applied for software quality modeling, such as Logistic Regression [32], Naive Bayes [33], and Decision Trees [34], [35]. Software measurement and defect data from a prior software release or similar project are used to train the software quality model, which is then applied to predict the quality of the target system with known software metrics. In the software industry, it is common for an organization to maintain several software metrics repositories for projects developed [36]. The data in these repositories are likely to follow similar patterns, especially if the organization enforces the same development life cycle, as well as the same coding and testing practices. While most existing related works focus on training using one software measurement data set, we emphasize including all relevant past projects during the training process. The working hypothesis is that multiple software repositories provide additional information that can improve the predictive performance of the trained software quality model. A common problem during software quality modeling is searching for an optimum model that adequately satisfies quality improvement goals. For example, the different costs of misclassifying fp and nfp modules poses model selection challenges. The search for an optimal solution is compounded when modeling with multiple software project data sets. In this section we review the current state of the art carried out by Yi (Cathy) Liu et al[37].

Software development includes prototype simulation, code inspection and measurement based analysis. Simply by identifying the errors software cannot be made perfect, analyzing the errors and then development is an important aspect .The most common solution is enhanced when manage multiple project datasets .NASA software metrics provides all the necessary data required for the same. In the thesis, two study cases of building software quality models are discussed. Genetic programming which was initially developed by M, Harman and B.Jones is the first one and the other non G.P. based techniques as such given by T.M.Khoshgooftarin his paper on software quality. The G.P. based technique follows Darwinian principle of survival .the case studies stated above are classified into 7 software measurements datasheets. The multiple repositories provide with some extra data that helps in development of software quality.

Software quality is next important aspect which is stressed in this paper. The software attributes such as the defect density and failure rate contribute to the development. Software metrics codes measurement schemes to build defect predictors and quality models. Classification of models is discussed that help to determine the quality of the software and is reliability. The software measurement data required is also available with the PROMISE engineering repositories. The following are the different data sets and software systems: kc1, kc2, kc3, cm1, mw1, pc1, jm1.with the help of case studies the thesis explains the development of softwares. The genetic programming process involves initialization, evaluation, selection, breeding and evolution. The tool used for the study is lilgp1.01 which was initiated by Douglas Zongker and Bill Punch. To represent the model performance and

complexity two fitness functions primary and secondary are used respectively. It was elicited that for higher values of modeling parameter correction of software would become easy.

In the non G.P. based technique, seventeen different machine learners are applied to the same seven software measurement datasets. They are as follows:

Classification technique	Acronym
Locally weighted learning	LWL Stump
1-Instance based learning	IB1
k-Instance based learning	IBk
Bagging	Bagging
Sequential minimal optimization	SMO
Logistic regression	LR
Ripple down rules	Ridor
One rule	OneR
Line of code	LOC
Partial decision tree	PART
Decision table	Decision table
WEKA's implementation c4.5	J48
Tree disc classification	TD
Alternating decision tree	ADTree
Repeated incremental pruning to produce error reduction	JRip
Random forest	Random forest
Naïve Bayes	NaiveBayes

From the results of the case studies, it was concluded that the G.P. based Baseline Classifier results for different merged training data sets. the kc1,kc2,kc3 results were relatively similar to their corresponding data set performances unlike mw1,pc1,jm1,cm1.how ever in the second study of G.P. based technique, the software quality development was based on summing up the validation phase. A single data set value was used for filtering error datasets.

From the non-G.P. based techniques, the average error rates for all 17 learners were determined individually. The test data was obtained based on the voting approach. it was states that jm1 has higher NECM values representing higher level of noise .the G.P. based technique is said to out beat the latter as the search space for the solution is wide when compared to the traditional classifications. The final structure of the solution is easier to determine in the G.P. bases technique. The empirical validity was also considered in this thesis. This technique was implemented with4 c projects, 2 c++ projects and one java project .the two additional metrics to capture the data set variation were introduced. With the help of other software metrics the software quality may be improved. An evolutionary process study was conducted in the best of the settings.

Observation: The paper focuses on the software development with multiple data repositories. The different G.P. based strategies were presented for the development process: baseline classifier, validation, validation-and-voting classifier. The validation-and-voting classifier is much better than baseline classifier since it is said to be less prone to over fitting and variance errors. The second case study tries to focus on this problem and indicates that the search based soft ware quality modeling approach produces better results when compared to the multiple software datasets. But with respect to the terms like quality, measurements and application domain, the software depositories are much similar to the target projects. This could provide a platform for object oriented metrics in the future study about quality development. By taking all other possible measurements it is also possible to improve the efficiency of the software.

XII. VALUE BASED SYSTEM QUALITY EVALUATION APPROACH

This section explores the value based system quality evaluation approaches and the discussion is centric to the proposal of the Frank Buschmann[38] that based on Big Ball Of Mud model.

Professionals have always been trying on software architectures and its support towards system qualities like flexibility, performance, and robustness. However, most of the software architectures use the Big Ball of Mud pattern which is a well structured system that is convenient for usage.

The Big Ball of Mud: To understand the Big Ball of Mud "theory", Some panelists said that nonfunctional quality is needed: user acceptance, maintenance-friendliness, and so on but another set said that good system quality is a result of using the "right" technologies, like COTS programming platforms. And mainly quality is costly, so to not waste precious time and budget, all design and realization efforts related to system quality should focus on aspects that directly contribute to the system's business success. Otherwise the system has no chance to get certified and no certification means hardly any use of it. Hence, quality with value is the finally convinced discussion followed by usage of sufficient system quality is necessary.

Value oriented Design: Software development projects invest a lot into achieving nonfunctional quality but, many project teams don't seem to know exactly the qualities needed by the system.

These qualities which are building blocks for system appear attractive from a marketing per-spective and that are hard to qualify and quantify even though any quality that isn't needed increases a system's architectural complexity and life-cycle costs; any quality that isn't provided with an appropriate degree decreases user acceptance. Consequently, the provided quality shares hardly any value to the system's success.

Pragmatic architects must determine the qualities a system needs and the level of quality needed and must also make corresponding design decisions that are needed to the quality requirements. If they observe scope creep or a tendency to neglect or overweight a system's quality demands, they should initiate a dialog with the relevant stakeholders to keep the project on track so that it refocuses on value-providing qualities. They should also communicate the identified quality demand's contribution to the system's business value and success, to get support from project sponsors for any necessary design and realization efforts. This above approach means that Big Ball of Mud

architectures are not bad. If requirements analysis reveals that we get the required value with a moderate level of quality, we don't need to design for the final solution; if the system's business success doesn't depend on a nonfunctional quality, Big Ball of Mud architecture is perfect.

The Architect's System Quality Toolbox is used to ensure a system provides valuable quality architects for methods, techniques, and practices that can be applied in the various activities of software development. In requirements engineering, the architect's interest lies in systems value being identified and narrowed as we find scenario-based requirements quite useful for fulfillment. Every scenario says about a specific user function.

From an architect' point of view, it is highly necessary to recognize the set of architectural quality needs as they have an effect on the system's concrete design. The finalized set of architecturally significant scenarios is the basic for guiding the concrete architecture. Two requirements for success are:

Focus on essence. The concept of walking skeletons is a baseline architecture and implementation that supports the functional requirements. We can evaluate the baseline and adjust explicitly and early in a project's life cycle, to attain its nonfunctional qualities.

And the other requirement is thoughtful design decision. Design tactics outline the solution space for a specific quality aspect in terms of potentially applicable patterns, practices and technologies using the specific qualities of the architectural scenarios. Design tactics support architects in selecting the "simplest solution that possibly could work." Thus they can create lean, economic, and elegant designs that maximize the value of the qualities addressed.

Observation: To get the expected operational and developmental quality, a test-driven approach is a powerful tool. Evaluations, simulations, and running code provide direct feedback as to about architecture and implementation support its quality requirements. Test-driven design is another vehicle to check the specified qualities are sufficient for providing the expected value or not. Unless the system is used, all its requirements, especially quality aspects, are just assumptions. Hence, you might need to adjust the assumed quality requirements when you see the system in action.

Basically, it's the responsibility of architects to act as the quality advocate of a system. Their part is very important, to ensure a system provides the right operational and developmental qualities and also to provide quality contributes measurable value.

XIII. CONCLUSION

Here, in this paper we discussed current state of the art in Software quality models. Our observations focused on various aspects of the proposed models usage those includes flexibility and reusability. We found from the above discussion that generalized SQA approaches are still in their initial stages. Many SQA models are suggested to achieve required goals but most of them are software development design specific. Thus, here we can conclude that there is a wide scope in research to develop generalized Software quality evaluation and assurance models those can generate Software quality evaluation test cases, which are specific to software development model selected.

REFERENCES

[1]B. W. Boehm, J. R. Brown, H. Kaspar, M. Lipow, G. J. Macleod, and M. J. Merrit. Characteristics of Software Quality. North-Holland, 1978.
[2]ISO. Software engineering – product quality – part 1: Quality model, 2001.
[3]D. Coleman, B. Lowther, and P. Oman. The application of software maintainability models in industrial software systems. J. Syst. Softw., 29(1):3–16, 1995.
[4]M. R. Lyu, editor. Handbook of Software Reliability Engineering. IEEE Computer Society Press and McGraw-Hill, 1996.
[5]S. Wagner. Using economics as basis for modelling and evaluating software quality. In Proc. First International Workshop on the Economics of Software and Computation (ESC-1), 2007.
[6]B. Kitchenham and S. L. Pfleeger. Software quality: The elusive target. IEEE Software, 13(1):12–21, 1996.
[7]E. Georgiadou. GEQUAMO—a generic, multilayered, cusomisable, software quality model. Software Quality Journal, 11:313–323, 2003.
[8]S. Khaddaj and G. Horgan. A proposed adaptable quality model for software quality assurance. Journal of Computer Sciences, 1(4):482–487, 2005.
[9]J.M¨unch and M. Kl¨as. Balancing upfront definition and customization of quality models. In Workshop-Band Software- Qualit¨atsmodellierung und -bewertung (SQMB 2008). Technische Universit¨at M¨unchen, 2008.
[10]M. Broy, F. Deissenboeck, and M. Pizka. Demystifying maintainability. In Proc. 4th Workshop on Software Quality (4-WoSQ), pages 21–26. ACM Press, 2006.
[11]F. Deißenb¨ock, S. Wagner, M. Pizka, S. Teuchert, and J.-F. Girard. An activity-based quality model for maintainability. In Proc. 23rd International Conference on Software Maintenance (ICSM '07). IEEE Computer Society Press, 2007.
[12]B. Kitchenham, S. Linkman, A. Pasquini, and V. Nanni. The SQUID approach to defining a quality model. Software Quality Journal, 6:211–233, 1997.
[13]V. Basili, P. Donzelli, and S. Asgari. A unified model of dependability: Capturing dependability in context. IEEE Software, 21(6):19–25, 2004.
[14]C. Frye. CMM founder: Focus on the product to improve quality, June 2008.
[15]N. Fenton. Software measurement: A necessary scientific basis. IEEE Trans. Softw. Eng., 20(3):199–206, 1994.
[16]N. E. Fenton and M. Neil. A critique of software defect prediction models. IEEE Trans. Softw. Eng., 25(5):675–689, 1999
[17]H.W.J. Rittel and M.M. Webber, "Dilemmas in a General Theory of Planning," Policy Sciences, vol. 4, no. 2, 1973, pp. 155–169.
[18]P. Abrahamsson et al., "New Directions on Agile Methods: A Comparative Analysis," Proc. 25th Int'l Conf. Software Eng. (ICSE 03), IEEE CS Press, 2003, pp. 244–254.
[19]D. Saff and M.D. Ernst, "An Experimental Evaluation of Continuous Testing during Development," Proc. ACM SIGSOFT Int'l Symp. Software Testing and Analysis, ACM Press, 2004, pp. 76–85.
[20]D.C. Schmidt, "Guest Editor's Introduction: Model- Driven Engineering," Computer, vol. 39, no. 2, 2006, pp. 25–31.
[21]G. Karsai et al., "Model-Integrated Development of Embedded Software," Proc. IEEE, vol. 91, no. 1, 2003, pp. 145–164.
[22]Hill, J.H.; Schmidt, D.C.; Edmondson, J.R.; Gokhale, A.S.; , "Tools for Continuously Evaluating Distributed System Qualities," Software, IEEE , vol.27, no.4, pp.65-71, July-Aug. 2010 doi: 10.1109/MS.2009.197,
[23]Koru, A.G.; Dongsong Zhang; El Emam, K.; Hongfang Liu; , "An Investigation into the Functional Form of the Size-Defect Relationship for Software Modules," Software Engineering, IEEE Transactions on , vol.35, no.2, pp.293-304, March-April 2009 doi: 10.1109/TSE.2008.90
[24]L.M. Ottenstein, "Quantitative Estimates of Debugging Requirements,"IEEE Trans. Software Eng., vol. 5, no. 5, pp. 504-514, 1979.
[25]J.R. Douceur, "The Sybil Attack," Proc. First Int'l Workshop Peer-to-Peer Systems, 2002.
[26]Limam, N.; Boutaba, R.; , "Assessing Software Service Quality and Trustworthiness at Selection Time," Software Engineering, IEEE Transactions on , vol.36, no.4, pp.559-574, July-Aug. 2010 doi: 10.1109/TSE.2010.2
[27]Hongyu Zhang; Sunghun Kim; , "Monitoring Software Quality

Evolution for Defects," Software, IEEE , vol.27, no.4, pp.58-64, July-Aug. 2010 doi: 10.1109/MS.2010.66
[28]Tarvo, A.; , "Mining Software History to Improve Software Maintenance Quality: A Case Study," Software, IEEE , vol.26, no.1, pp.34-40, Jan.-Feb. 2009 doi: 10.1109/MS.2009.15
[29] L.C. Briand, W.L. Melo, and J. Wust, "Assessing the Applicability of Fault-Proneness Models across Object-Oriented Software Projects," IEEE Trans. Software Eng., vol. 28, no. 7, pp. 706-720, July 2002.
[30] N.J. Pizzi, R. Summers, and W. Pedrycz, "Software Quality Prediction Using Median-Adjusted Class Labels," Proc. IEEE CS Int'l Joint Conf. Neural Networks, vol. 3., pp. 2405-2409, May 2002.
[31] A. Koru and H. Liu, "Building Effective Defect-Prediction Models in Practice," IEEE Software, vol. 22, no. 6, pp. 23-29, Nov./Dec. 2005.
[32] N.F. Schneidewind, "Investigation of Logistic Regression as a Discriminant of Software Quality," Proc. IEEE CS Seventh Int'l Software Metrics Symp., pp. 328-337, Apr. 2001.
[33] T. Menzies, J. Greenwald, and A. Frank, "Data Mining Static Code Attributes to Learn Defect Predictors," IEEE Trans. Software Eng., vol. 33, no. 1, pp. 2-13, Jan. 2007.
[34] L. Guo, B. Cukic, and H. Singh, "Predicting Fault Prone Modules by the Dempster-Shafer Belief Networks," Proc. IEEE CS 18th Int'l Conf. Automated Software Eng., pp. 249-252, Oct. 2003.
[35] T.M. Khoshgoftaar and N. Seliya, "Comparative Assessment of Software Quality Classification Techniques: An Empirical Case Study," Empirical Software Eng. J., vol. 9, no. 3, pp. 229-257, 2004.
[36] N.E. Fenton and S.L. Pfleeger, Software Metrics: A Rigorous and Practical Approach, second ed. PWS Publishing, 1997.

AUTHOR PROFILE

N.Rajasekhar reddy was born in Madanapalli, February 28.He was received Bachelor's degree in Computer Science in S.V University and M.Tech degree from Satyabama University respectively. After working as a research assistant and an assistant professor in the Dept. of ComputerScience and Engineering, Madanapalli Institute of Technology and Science, Andhra Pradesh, India. His research interest includes Software Engineering, Software Quality Assurance and Testing. He was published 4 international journal papers and 5 National journal papers in Software Engineering. He is a member of SCIE, ISTE, and IEEE.

R.J.Ramasree was born in Tirupati and received M.S degree in Bits Pilani and Doctoral degree in Computer Science, S.V University respectively. Then she was working as assistant professor in the Dept. of Computer Science in Rastriya Sanskrit Vidya peeta University. After professor in the Faculty of Computer Science, Raastriya Sanskrit VidyaPeeta, Tirupati. . Her research interest includes Data Mining.