A Panoramic Approach On Software Quality Assurance Proposed By CMM And XP

CH.V. Phani Krishna[*1], G.Rama Krishna[*2] and K.Rajasekhara Rao[*3]

[1]Associate professor, CSE Department, KL University, Guntur dt., India.

[2]professor, CSE Department, KL University, Guntur Dt., India

[3]Dean of student and faculty welfare, KL University, Guntur Dt., India.

Abstract
The main objective of this paper is to compare Capability Maturity Model (CMM) and Extreme Programming (XP) regarding their software quality support in terms of software quality development. The main goal is to analyze or measure how the code is framed for particular software, and apply software to show the result.

Key words:
Sqa,Xp,Cmm.

1. Introduction:

The software quality engineering focuses on the processes involved in the development and establishment of software quality. Software quality engineering includes software quality development and software quality assurance. Software quality development consists of requirements engineering, system and software design and implementation. Software quality assurance consists of software quality assurance, quality management and verification and validation. Software quality is achieved by three approaches: testing and static analysis and development approaches. The integration of all three approaches is the most desirable approach.

Different users think differently about the quality of software. The end-user expects the software to help him to do the job faster and easier with adequate help. The buyer expects the software to meet the specifications within the contract terms. The developer attempts to trace defects and focuses faster development as well as higher productivity. The maintainer expects software to be understandable, testable, and modifiable, with all documentation.

The characteristics of software quality in product transition are reusability, portability and interoperability. The characteristics of software quality in product revision are maintainability, adaptability and expandability. The characteristics of software quality in product operation are usability, security, efficiency, correctness and reliability. The attributes of software quality are manageability, efficiency, safety, expandability, reliability, flexibility and usability.

There are quantitative as well as qualitative benefits in maintaining quality assurance. The Quantitative benefits are reduced costs, greater efficiency, better performance, less unplanned work and fewer disputes. The Qualitative benefits are improved visibility and predictability, better control over contracted products, improved customer confidence, better quality, problems show up earlier and reduced risk.

2. Software Quality Assurance Proposed by CMM:

It is well known the CMM describes an evolutionary improvement path to a mature disciplined process.

CMM defines key practices to improve the ability of the organization to meet goals for cost, functionality and quality. SQA activities are defined at level 2

According to CMM the purpose of software quality assurance (SQA) is to provide the management with appropriate visibility into the process being used by the software project and of the products being built. It is required that the project follows a return organizational policy for implementing the SQA.

CMM defines eight activities to be performed as follows:

- A SQA plan is prepared for the software project according to documented procedure.
- SQA's group activities includes:
- Responsibilities and authority of SQA group
- Resource requirements of SQA group
- Schedule and funding of the project.
- Participation in establishing the software development plan (SDD).
- Evaluations to be performed.
- Audits and reviews to be conducted.
- Projects standards and procedures forming basis for SQA reviews.

Manuscript received February 5, 2013
Manuscript revised February 20, 2013

- Procedures for documenting and tracking non-Compliance issues.
- Documentation to produce.
- Method and frequency to provide feedback to other related group.
- The SQA group participates in the preparation and review of the project's software development plan, standards and procedures and audit the software project.
- The SQA group audits designated software work products to verify compliance.
- The SQA group periodically reports the result of its activities to the software engineering group.
- Deviations identified in the software activities and software work products are documented and handled according to documented procedure.
- The SQA group conducts periodic reviews of its activity and findings with customers SQA personnel as appropriate.

3. CMM levels key process areas and their purpose:

3.1 Initial:

This is the starting point for use of a new or undocumented, repeated process. Little documentation is necessary if any processes and procedures take place. Success is only achieved by the heroic actions of team members.
When to use:
Used for a kind projects of very limited scope.

3.2 Repeatable:

The process is at least documented sufficiently such that repeating the same steps may be exempted. Enough documentation exists that the QA process is repeatable.
When to use:
This is used for any project that will be done again, whether as an upgrade or a somewhat similar variation.

3.3 Defined:

The process is defined/confirmed as a standard business process, and decomposed to levels 0, 1 and 2 (the latter being Work Instructions).QA documentation and processes & procedures are standardized. Templates exist for all documentation and a QA "system" exists.
When to use:
This is critical for a QA department that must provide QA for multiple projects. This avoids reinventing the wheel for each project.

3.4 Managed:

The process is quantitatively managed in accordance with agreed-upon metrics. The exact time & resources required to provide adequate QA for each product is known precisely so that timetables and quality levels are met consistently.
When to use:
This requires an existing data set based on previous QA projects. This level can only be achieved by well documented experience.

3.5 Optimizing:

Process management includes deliberate process optimization / improvement. QA processes and procedures are understood well enough to be refined and streamlined.
When to use:
This should be actually used in every stage. In Level 5, this is the only thing left to work on.
It would be enlightening to conduct a CMM assessment of a team successfully practicing XP. In fact, XP team would achieve a maturity level 2 or better. CMM level 2 is about managing project requirements and schedules effectively and repeatedly. XP claims to do just that, using story cards and a planning game [4].
Thus, the software engineering goals are worthy and they can even be implemented with lightweight methodologies where appropriate. XP is compatible to CMM as well. Software quality assurance consists of Software quality assurance, quality management and verification and validation [5]. Software quality is achieved by three approaches: Testing, Static analysis and development approach. The integration of all the three approaches is the most desirable approach. A different categorization of approaches towards software quality regards four ways to establish software quality: Software quality via better quality evaluation, better measurement, better processes and better tools [6].
Large-scale quality models like Capability Maturity Model (CMM) or ISO-9001 tend to form a SQA in terms of a "process police". [7] SQA takes care only that the process requirements are met but does not consider the quality of the process itself. Instead of SQA in terms of CMM or ISO 9001 a better solution is to embed quality evaluation in the development process.
XP require certain adaptations in order to fulfill CMM requirements specialized maturity models for XP are introduced by combining Capability Maturity Model (CMM) with Personal Software Process (PSP) [8, 3]. Therefore, instead of eliciting SQA in terms of CMM a better solution can be embedded for quality evaluation in XP [9, 10].

4. Software Quality Assurance Proposed by XP:

4.1 Iterative Software Development:

To establish higher software quality, a software development process has to use an iterative and incremental development approach. By using iterative approach a process can gain more flexibility in dealing with changing requirements or scope. The Short Releases of the product force early feedback from the customer as well as stakeholders which is important for improvement of overall quality of the software. XP builds on a very strict iterative approach limiting the time needed to encounter errors and forces developers to fix the problem as soon as possible.

4.2 Quality as a Primary Objective:

XP software development process defines quality as a major objective to improve the overall quality of the software. Quality targets have to be defined by involving project team members and customer (On-Site Customer). Thus the quality goals become achievable and measurable.

4.3 Continuous Verification of Quality:

This includes extensive testing. Besides internal unit testing, external acceptance tests with the customer are needed too, in order to verify that the product fulfills the needs and requirements of the customer (Test-Driven Development).

4.4 Customer Requirements:

The requirements of the customer who normally does not have a deep technical knowledge have to be considered, so that developers are able to build an application based on that information. Thus it is necessary that the project team understands the customer and his business. Otherwise it is not possible to implement the customer needs accurately. XP teams focuses on the customer needs and requirements throughout the entire project by means of communication and by framing user stories.

4.5 Architecture Driven:

Architecture of a system has a major impact on the overall quality of the product. Using a simple well-designed architecture allows easy integration and reuse (Simple Design and Continuous Integration).

4.6 Focus on Teams:

Focusing on team work also effects the motivation of project members. Seeing everyone as an equally important part of the project leads to a high identification of the team members with the product. Hence the project code is not owned by any single programmer but owned by the team collectively (Collective Code Ownership).

4.7 Pair Programming:

Better solutions are more likely with Pair Programming since two persons most likely have different perspectives of the same problem and therefore they complement each other in solving it. This approach saves time and minimizes the number of errors. This is an explicit practice of XP.

4.8 Tailoring with Restrictions:

Software development process should rely on core elements. Building on these core elements the process should adapt practices (tailoring) according to the project type and project size (eg. RDP)

4.9 Risk management:

Risk management enables early risk mitigation and the possibility to act instead of to react to problems and risks. A well-defined risk awareness and mitigation management form together an effective risk management and is a key factor in achieving high product quality.

5. Existing System

In the existing system, a large number of codes are divided into only two modules. So in the existing system, performance analysis takes more time and is also not accurate.

As per Mancoridis et al., the earliest of software metrics deal with the measurement of code complexity and its maintainability. He measured the Modularization Quality (MQ) which is the combination of coupling and cohesion. Cohesion is measured as the ratio of the number of internal function-call dependencies that actually exist, to the maximum possible internal dependencies. Coupling is measured as the ratio of the number of actual external function-call dependencies between the two subsystems, to the maximum possible number of such external dependencies. The system level MQ is calculated as the difference between the average cohesion and the average coupling.

6. Process of Proposed System:

In the proposed system, we have considered the leaf nodes of the directory hierarchy of the original source code to be the most fine-grained functional modules. All the files (and functions within) inside a leaf level directory are considered to belong to a single module, with the module corresponding to the directory itself. In this manner, all leaf level directories form the module set for the software.

- A lot of work has been done in the past on automatic approaches for code reorganization. There are certain principles, which are most applicable to code reorganization. Our current ongoing effort is targeted on the reorganization of legacy software, containing millions of lines of non-object oriented code. This code was never modularized, or the modularization was very poor. The problem could be attributed as reorganization of millions of lines of code into modules. This code could reside in thousands of files, in hundreds of directories. Here, each module is formed by grouping a set of entities like files, functions, data structures and variables into a logically interconnected unit.
- Modularization is based on certain design principles:

Principle1: Principles Related to Similarity of Purpose

A module is a cluster of a set of data structures and functions that together offer a distinct purpose. To rephrase, the structures used for representing knowledge and any associated functions in the same module should fit together on the basis of similarity-of-service as opposed to, for instance, on the basis of function call dependencies. Clearly, every service is related to a specific purpose. The following principles are presented as coming under the "Similarity of Purpose" rubric:

Maximization of Module Coherence on the Basis of Similarity and Singularity of Purpose.

- Minimization of Purpose Dispersion
- Maximization of Module Coherence on the Basis of Commonality of Goals
- Minimization of Goal Dispersion.

Principle 2: Principle Related to Module Compilability

- A universal basis of inter module compilation dependency is that a file from one module needs, through import or include declarations, one or more files from a different module. As software systems evolve and some modules seem like utilities to developers, it is very easy for such interdependencies to become circular. For apparent reasons, these compilation inter-dependencies make it difficult for modules to grow in parallel, and be tested independently. Hence, as far as possible, it must be possible to compile each module independently of the other modules.

Principle 3: Principle Related to Module Extendibility

One of the most important reasons for object-oriented software development is that the classes can be easily extended whenever one wants a more specialized functionality. Extending object-oriented software through the idea of sub-class allows for a more ordered approach to software development and maintenance, since it makes code authorship and its responsibility easy to identify. While module- level compartmentalization of code does not follow the types of software extension rules that are easy to implement in object-oriented approaches, one nevertheless wants the modules to have similar properties when it comes to code extension and enhancement. The following principle takes into account these aspects of code modularization:

- Maximization of the Stand-Alone Module
- Extendibility

Principle 4: Principle Related to Module Testability

Testing is a vital part of software development. At the most, testing must make sure that software conforms to the existing standards and protocols. This kind of testing is mostly called requirements-based testing. But, most important, testing must guarantee that the software code must act as expected for a whole variety of inputs, both correct and incorrect, and at multiple levels. These levels constitute the level of program at the individual function, and at module interactions level. Testing must account for variety of competencies of all causes that interact with the software. Testing procedures can encounter combinatorial problems if the modules cannot be tested independently. This means that if each module is tested for X inputs, then two inter-dependent modules need to be tested for X2 inputs. A modularization procedure must adhere to accomplish the following principle:

- Maximization of the Stand-Alone Testability of Modules

Principle 5: Principles Related to Module Size

When a new software development is started afresh, one cannot have all the modules to be of the same size, and equal to some pre-decided number. Nevertheless, when the modularizing legacy code is completely unorganized, it is essential to be able to bias a clustering algorithm to produce modules of approximately the same size, and whose value depend on considerations which are related to software maintenance.

Putting the whole code in a single module is theoretically a correct modularization, though not a useful one. Hence, we need metrics that can maneuver a modularization algorithm away from making very large modules, towards making modules in the same size, while at the same time also ensure that other considerations are not violated. The following two principles deal with this necessity:

- Principle of Observance of Module Size Bounds
- Principle of Maximization of Module Size

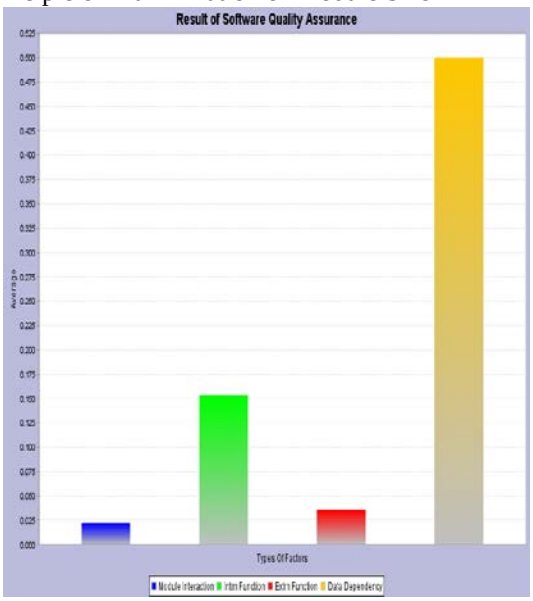

FIG1. Result of Software Quality Assurance by CMM

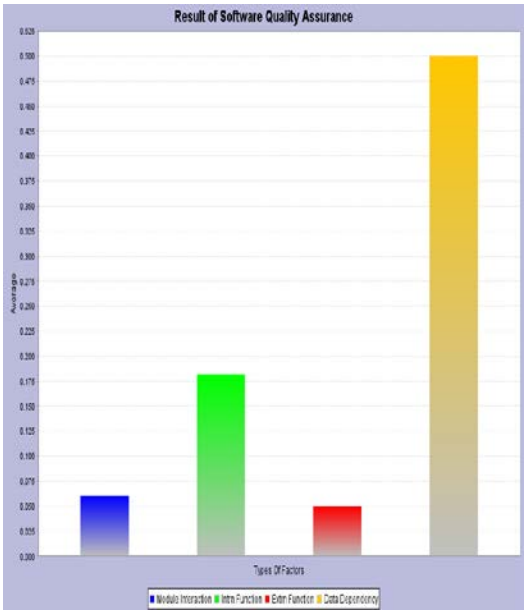

FIG2. Result of Software Quality Assurance by XP

Conclusion:

Thus, Practices of XP support software quality development as well as software quality assurance. XP require certain adaptations in order to fulfill CMM requirements specialized maturity models for XP are introduced by combining Capability Maturity Model (CMM) with Personal Software Process. However, much software quality support is implicitly present in XP principles.

References:

[1] B.W.Boehm. Software Engineering Economics. Prentice Hall, Englewood Cliffs, NJ, 1981.

[2] Ward, W.A., and Venkataraman.B, Some observations on Software quality, in proceedings of the 37th annual southeast regional conference (CD-ROM), ACM, 1999, Article No.2.

[3] Microsoft Cooperation: Microsoft Solutions Framework White Paper, Microsoft Press, 1999.

[4] Huo, M., Verner, J., Zhu, L., Babar, M.A: Software quality and agile methods. In proceedings of COMPSAC 04, IEEE Computer Soc., 2004, pp.520-25.

[5] Paulk, N.C: Extreme Programming from a CMM Perspective. IEEE software, vol. 18, no.6, IEEE, Nov-Dec.2001, pp.19-26.

[6] Nawrocki,J.,Walter, B.,and Wojciechowski, A.: Toward maturity model for Extreme Programming: In proceedings Euromicro Conference, 2001.IEEE,2001,pp. 233-9.

[7] Baker, E.B., Which way, SQA? .IEEE-Software, vol.18, no.1; Jan.-Feb. 2001; pp. 16-18.

[8] ManZoni, L.V.; Price, R.T.: identifying extensions required by RUP(Rational Unified Process) to comply with CMM (Capability Maturity Model) level 2 and 3. IEEE Transaction on Software Engineering, Vol 29, no.2, IEEE, Feb.2003,pp.181-192.

[9] Pollice, G.: Using Rational Unified Process for small Projects: Expanding Upon Extreme Programming. A Rational Software White Paper, Rational, 2001.

[10] Runeson, P., Isacsson, P.:Software Quality Assurance Concepts and Misconceptions, In Proceedings of the 24th EUROMICRO Conference, IEEE Computer Soc, 1998, pp.853-9.

[11] Osterweil, L.J.: Improving the quality of software quality determination processes, In the Proceedings of the IFIP TC2/WG2.5 Working Conference on Quality of Numerical Software. Assessment and Enhancement, Chapman & Hall, London, 1997, pp.90-105.

Ch.V.Phani Krishna is an Associate Professor in Computer Science and Engineering at KL University. Having more than 10 years of teaching and research experience, he is actively engaged in the research related to Software Engineering. He published 14 International journals. Having Life Membership of ISTE, CSI, IACSIT.

K.RAJASHEKARA RAO is a Professor of Computer Science and Engineering at KL University and presently holding several key positions in KL University, as Dean (Faculty & Student Affairs) & Principal, KL College of Engineering (Autonomous).Having more than 25years of teaching and research experience, Prof. Rao is actively engaged in the research related to Embedded Systems, Software Engineering and Knowledge Management. He had obtained Ph.D in Computer Science & Engineering from Acharya Nagarjuna University (ANU), Guntur, Andhra Pradesh and produced 35 publications in the International/National Journals and Conferences.

He has been adjudged as best teacher and has been honored with "Best Teacher Award", six times. Dr. Rao is a Fellow of IETE, Life Member's of IE, ISTE, ISCA & CSI (Computer Society of India). He has been the past Chairman of the Koneru Chapter of CSI. Presently, Prof. K.R.Rao is the CSI State Student Coordinator of Andhra Pradesh.