Chapter 9

QUALITY MANAGEMENT

-Amir, Sudeep Rawal & Shristi Maharjan

9.1 Quality concept

With development of software it is necessary to develop a concept. How might a software development organization need to control variation? From one project to another, we want to minimize the difference between the predicted resources needed to complete a project and the actual resources used, including staff, equipment, and calendar time. In general, we would like to make sure our testing program covers a known percentage of the software, from one release to another. Not only do we want to minimize the number of defects that are released to the field, we'd like to ensure that the variance in the number of bugs is also minimized from one release to another. (Our customers will likely be upset if the third release of a product has ten times as many defects as the previous release.) We would like to minimize the differences in speed and accuracy of our hotline support responses to customer problems. The list goes on and on.

9.2 Quality Assurance

The terms quality assurance is widely used in manufacturing industry. Quality assurance (QA) is the definition of process and the standards that should lead to high quality products and introduction of quality process into manufacturing process

In software industry, different companies and industry sectors quality assurance is used in different ways; sometimes Q/A simply means the definition of procedures, processes and standards that are aimed at ensuring software quality is achieved. In other cases Q/A also includes all configuration, management, verification and validation activities that are applied after a product has been handed over by a development team.

The QA team in most companies is responsible for managing the release testing process. This means they manage the testing of the software before it is released to customers.

Software Quality Assurance encompasses

- A quality management approach,

- Effective software engineering technology (methods and tools),
- Formal technical reviews that are applied throughout the software process,
- A multitiered testing strategy,
- Control of software documentation and the changes made to it,
- A procedure to ensure compliance with software development standards (when applicable), and
- Measurement and reporting mechanisms.

9.3 SOFTWARE REVIEWS

Software reviews is a “filter” for the software process. This are applied at various points during software engineering and serve to uncover errors and defects that can be removed. Software reviews purify software engineering activities we have called analysis, design and coding,

Reviews can be done in different ways. An informal meeting around a coffee shop is a form of review, if technical problem are discussed. A formal presentation of software design to an audience of customers management and technical staffs is also form of review. A formal technical review is the most effective filter form a quality assurance point.

9.4 FORMAL TECHNICAL REVIEWS

A formal technical review is software quality activity performed by software engineers for following objectives;

- To uncover error in function, logic or implementation for any representation of the software.
- Verify software if it has met is requirement.
- Ensure that software has been represented according to predefined standards.
- To achieve software that is developed in uniform manner.
- To make projects more manageable.

In addition FTR serves as a training ground for junior engineers to observe different approach of software analysis, design and construction. The FTR also serves to promote backup & continuing because number of people becomes familiar with parts of the software. Each FTR is conducted as meeting and will be successful only if properly planned, controlled, and attended.

Review Meeting:

Regardless of the FTR format that is chosen every meeting should abide by following constraints:

- Between 3 to 5 people should be involved in review meeting
- Advance preparation should occur but should be less than 2 hours of work for each person
- The duration of the review meeting should not be more than 2 hours

Given these constrains it should be obvious that an FTR focuses on a specific part of the overall software for example rather than attempting to review an entire design, walkthroughs are conducted for each components the individuals who has developed the work product – the producers- informs the project leader that the work product is complete and a review is required. The project leader contacts a review leader who evaluates the product for readiness, generates copies of product materials and distributes it to two or three reviewers for advance preparation. The reviewer including review leader reviews the product and establishes an agenda for the review meeting .After the review meeting one of the reviewer records the valid problems or errors that are discovered.

At the end of the reviews all the attendees of FTR must decide whether to

1. Whether to accept the product without further modification.
2. Reject the product due to several errors.
3. Accept the product provisionally (with minor errors that could be correct but further additional review is not required.

Review Reporting and Record Keeping

During the FTR, a reviewer (the recorder) actively records all issues that have been raised. In addition a FTR summary report is completed which answers 3 questions:

- Who was reviewed?
- Who reviewed it?
- What were the findings and conclusions?

The review issue is a single page form. The review issue serves two purposes:

1. To identify the problem areas within the product.
2. To serve an action item check list that guides the producer as correctness are made.

Review Guidelines

Guidelines for conducting formal technical reviews must be established in advance all reviews, agreed upon and then followed:

1. Review the product not the producer:
   - ➢ FTR should leave warm feeling of accomplishment.
   - ➢ Errors should be pointed out gently.
   - ➢ Tone of meeting should be loose and constructive.
2. Set an agenda and maintain it.
   - ➢ FTR must be kept on track and on schedule
   - ➢ FTR must be responsive and shouldn't be afraid to nudge people.

3 Limit debate and rebuttal

- ➢ When an issue is raised there may be universal agreement. Rather than spending time debating the question, the issue shouldn't be recorded for further discussion.

4 Enunciate problems but don't try to solve every problem noted:

Review is time for discussing related problems rather than solving problems instantly.

5 Take written notes:

It is good to make notes on wall board so that wording and priorities can be assessed by other reviewers;.

6 Limit the number of participants and insist upon advance preparation:

Number of people must be minimum and review team member must be preparing in advance.

7 Develop a checklist for each product that is likely to be reviewer:

A checklist helps the review leader to structure the FTR meeting and helps each to focus on important issues.

8 Conduct meaningful training for all review:

For effective FTR all participants should receive training. The training should stress both issues and human Psychological side of review.

9 Review your early review:

Debriefing can be beneficial in uncovering problems with the review process itself.

9.5 FORMAL APPROACHES TO SQA

Over past 2 decades software engineering community has argued for more formal SQA is required. A strict and detailed syntax and semantics (meaning of words and sentences) can be defined for every programming language and a strict and detailed approach to the specification of the software requirement is required, if requirement and programming language can be represented in strict and detailed manner then we can apply mathematical proof of correctness to demonstrate that a program conforms exactly to its specifications.

To prove programs correctness we can use Dijkistra and linger, mills and Witt etc. proofs.

9.6 STATISTICAL SOFTWARE QUALITY ASSURANCE

Statistical is quality assurance reflects a growing trend throughout industry to become more quantitative about quality. For software statistical quality assurance implies the following steps.

- Information about software defects is collected and categorized.
- An attempt is made to trade each defect to its underlying cause.
- Using the Pareto principle, isolate 20percent.
- Once the vital few causes have been identified move to correct the problems that have caused the defects.

9.7 Software reliability

Ability: The probability that the system, at a point in time, will be operational and able to deliver the requested service.

Reliability: The probability of failure free operation over a specific time, in a given environment, for a specific purpose.

Software reliability is not independent. Hardware failure can generate spurious signals that are outside the range of inputs expected by the software. The software can then behave unpredictably and produce unexpected outputs. These may confuse and consequently stress the system operator. Failure is non-conformance to the software requirement.

For reliability testing, data is gathered from various stages of development, such as the design and operating stages. The tests are limited due to restrictions such as cost and time restrictions. Statistical samples are obtained from the software products to test for reliability of the software. Once sufficient data or information is gathered, statistical studies are done. Time constraints are handled by applying fixed dates or deadlines for the tests to be performed. After this phase, design of the software is stopped and the actual implementation phase starts. As there are restrictions on costs and time the data is gathered carefully so that each data has some purpose and gets its expected precision. To achieve the satisfactory results from reliability testing one must take care of some reliability characteristics. For example Mean Time to Failure (MTTF) is measured in terms of three factors

1. operating time,
2. number of on off cycles,
3. Calendar time.

If the restrictions are on operation time or if the focus is on first point for improvement, then one can apply compressed time accelerations to reduce the testing time. If the focus is on calendar time (i.e. if there are predefined deadlines), then intensified stress testing is used.

Measurement of reliability in **Mean Time between Failures (MTBF)**
MTBF=MTTF+MTTR
Where,
MTTF: Mean time to failure
MTTR: Mean time to repair

Ability= $\frac{MTTF}{MTTF+MTTR}$***100**

Reliability= $\frac{MTTF}{1+MTBF}$

Over 225 models have been developed since early 1970s, but how to quantify software reliability still remains unsolved. There is no single model which can be used in every situation. There is no model which either complete or fully developed.
Many software models contain:

1. Assumptions
2. Factors
3. Mathematical function

Software reliability can be divided into categories:
a. Prediction modeling
b. Estimation modeling

These modeling techniques follow observation and analyzes with statistical inference.

Prediction Model This model uses historical data. They analyze previous data and some observations. They usually made prior development and regular test phases. The model follow the concept phase and the predication from the future time.

Estimation Model Estimation model uses the current data from the current software development effort and doesn't use the conceptual development phases and can estimate at any time.

9.8 A framework for software metrics

SOFTWARE METRIC IS A MEASURE OF SOME PROPERTY OF A PIECE OF SOFTWARE OR ITS SPECIFICATIONS. GOOD QUALITY, RELIABILITY AND MAINTAINABILITY ARE IMPORTANT ATTRIBUTES OF ENTERPRISE APPLICATIONS AND HAVE A HUGE IMPACT ON THE SUCCESS ON THE ECONOMICS OF THE BUSINESSES POWERED. IT REALLY, REALLY MATTERS.

A Framework for Software Metrics

Measures, Metrics, and Indicators. These three terms are often used interchangeably, but they can have subtle differences

- Measure :Provides a quantitative indication of the extent, amount, dimension, capacity, or size of some attribute of a product or process
- Measurement: The act of determining a measure
- Metric:(IEEE) A quantitative measure of the degree to which a system, component, or process possesses a given attribute
- Indicator: A metric or combination of metrics that provides insight into the software process, a software project, or the product itself

Activities of a Measurement Process

- Formulation: The derivation (i.e., identification) of software measures and metrics appropriate for the representation of the software that is being considered
- Collection: The mechanism used to accumulate data required to derive the formulated metrics
- Analysis: The computation of metrics and the application of mathematical tools.
- Interpretation: The evaluation of metrics in an effort to gain insight into the quality of the representation.

- Feedback:Recommendations derived from the interpretation of product metrics and passed on to the software development team

–

9.9 Matrices for analysis and design model

Metrics for the analysis

- Functionality delivered
  - Provides an indirect measure of the functionality that is packaged within the software
- System size
  - Measures the overall size of the system defined in terms of information available as part of the analysis model
- Specification quality
  - Provides an indication of the specificity and completeness of a requirements specification

Metrics for the Design Model

- Architectural metrics
  - Provide an indication of the quality of the architectural design
- Component-level metrics
  - Measure the complexity of software components and other characteristics that have a bearing on quality
- Interface design metrics
  - Focus primarily on usability
- Specialized object-oriented design metrics
  - Measure characteristics of classes and their communication and collaboration characteristics

9.10 ISO Standard

ISO 9126 is an international standard for the evaluation of software quality.

The standard is divided into four parts, which address, respectively, the following subjects: quality model; external metrics; internal metrics; and quality in use metrics.

Quality Mode The quality model established in the first part of the standard, ISO 9126-1, classifies software quality in a structured set of characteristics and sub-characteristics. These are also considered as nonfunctional requirements metrics. These are:

1-Functionality - A set of attributes that bear on the existence of a set of functions and their specified properties. The functions are those that satisfy stated or implied needs.

- Suitability
- Accuracy
- Interoperability – the capability of different programs to exchange data via a common set of exchange formats, to read and write the same file formats, and to use the same protocols
- Compliance – the flexibility of the software to accept new features and enhancements.
- Security – preventing unauthorized access to the software.

2- Reliability - A set of attributes that bear on the capability of software to maintain its level of performance under stated conditions for a stated period of time.

- Maturity
- Recoverability - software product can be modified in order to correct defects, meet new requirements, make future maintenance easier, or cope with a changed environment
- Fault Tolerance - the property that enables a system (often computer-based) to continue operating properly in the event of the failure of (or one or more faults within) some of its components.

3- Usability - A set of attributes that bear on the effort needed for use, and on the individual assessment of such use, by a stated or implied set of users.

- Learnability
- Understandability
- Operability - ability to keep a system in a functioning and operating condition.

4- Efficiency - A set of attributes that bear on the relationship between the level of performance of the software and the amount of resources used, under stated conditions.

- Behavior
- Resource Behavior

5- Maintainability - A set of attributes that bear on the effort needed to make specified modifications.

3. Stability
4. Changeability
5. Testability

6- Portability - A set of attributes that bear on the ability of software to be transferred from one environment to another.

6. Installability
7. Replaceability
8. Adaptability

9.11 CMM

Capability Maturity Model Integration (CMMI)
is a process improvement approach. CMMI can be used to guide process improvement across a project, a division, or an entire organization. Processes are

rated according to their maturity levels, which are defined as: Initial, Repeatable, Defined, Quantitatively Managed, and Optimizing. CMMI currently addresses three areas of interest:

1. Product and service development — CMMI for Development (CMMI-DEV),
2. Service establishment, management, — CMMI for Services (CMMI-SVC), and
3. Product and service acquisition — CMMI for Acquisition (CMMI-ACQ).

CMMI was developed by a group of experts from industry, government, and the Software Engineering Institute (SEI) at Carnegie Mellon University. CMMI models provide guidance for developing or improving processes that meet the business goals of an organization. A CMMI model may also be used as a framework for appraising the process maturity of the organization. By January of 2013, the entire CMMI product suite was transferred from the SEI to the CMMI Institute, a newly created organization at Carnegie-Mellon.

CMMI originated in software engineering but has been highly generalized over the years to embrace other areas of interest, such as the development of hardware products, the delivery of all kinds of services, and the acquisition of products and services. The word "software" does not appear in definitions of CMMI. This generalization of improvement concepts makes CMMI extremely abstract. It is not as specific to software engineering as its predecessor, the Software CMM.

9.12 SQA plan

The SQA Plan provides a road map for instituting software quality assurance. Developed by the SQA group, the plan serves as a template for SQA activities that are instituted for each software project. A standard for SQA plans has been recommended by the IEEE.

Initial sections describe the purpose and scope of the document and indicate those software process activities that are covered by quality assurance. All documents noted in the SQA Plan are listed and all applicable standards are noted. The management section of the plan describes SQA's place in the organizational structure, SQA tasks and activities and their placement throughout the software process, and the organizational roles and responsibilities relative to product quality.

The documentation section describes (by reference) each of the work products produced as part of the software process. These include:

• Project documents (e.g., project plan)
• Models (e.g., ERDs, class hierarchies)
• Technical documents (e.g., specifications, test plans)
• User documents (e.g., help files)

In addition, this section defines the minimum set of work products that are acceptable to achieve high quality. The standards, practices, and conventions section lists all applicable standards and practices that are applied during the software process (e.g., document standards, coding standards, and review guidelines). In addition, all project, process, and (in some instances) product metrics that are to be collected as part of software engineering42 work are listed.

The reviews and audits section of the plan identifies the reviews and audits to be conducted by the software engineering team, the SQA group, and the customer. It provides an overview of the approach for each review and audit.

The test section references the Software Test Plan and Procedure. It also defines test record-keeping requirements. Problem reporting and corrective action defines procedures for reporting, tracking, and resolving errors and defects, and identifies the organizational responsibilities for these activities. The remainder of the SQA Plan identifies the tools and methods that support SQA activities and tasks; references software configuration management procedures for controlling change; defines a contract management approach; establishes methods for assembling, safeguarding, and maintaining all records; identifies training required to meet the needs of the plan; and defines methods for identifying, assessing, monitoring, and controlling risk.

9.13 Software certification

Software certification demonstrates the reliability and safety of software systems in such a way that it can be checked by an independent authority with minimal trust in the techniques and tools used in the certification process itself. It builds on existing software assurance, validation, and verification techniques but introduces the notion of *explicit software certificates*, which contain all the information necessary for an independent assessment of the demonstrated properties.

Software certification comprises a wide range of formal, semi-formal, and informal assurance techniques, including formal verification of compliance with explicit safety policies, system simulation, testing, code reviews and human "sign offs", and even references to supporting literature. Consequently, the certificates can have different types, and the certification process requires different mechanisms.

Certificates A certificate contains all information necessary for an independent assessment of the properties claimed for an artifact. At its most abstract, a certificate thus has to represent the three entities involved in the certification process,

(i) the artifact being certified,

(ii) (ii) the property being asserted, and

(iii) (iii) The certification authority.

Certifiable Artifacts Certifiable artifacts include not only the conventional software artifacts (e.g., product families, completed systems, individual components, or even code fragments) but also supporting non-software artifacts: requirements documents, system designs, component specifications, test plans, individual test cases, scientific and engineering data sets, and others.

Certificate Hierarchies The certificates for an artifact are not an unstructured collection but exhibit some hierarchical structure. This structure is determined by two independent dimensions,

(i) The system structure, and

(ii) The certificate types.

The internal structure of a *system* is reflected in the certificate hierarchy.

Certifiable Properties and Certification Authorities Traditional V&V has only addressed a restricted range of formal properties. Realistically, however, software development requires a wide range of notions of software reliability, safety, and validity, each with an appropriate certification authority. This must all be supported by a customizable SCMS.

Release Policies In the context of certification, a release refers to the transition of an artifact into a new defined state: for example, launch, system integration test, alpha and beta testing phases, spiral anchor-point milestones, or code inspection. A release policy formally describes under which conditions an artifact is deemed to be in an adequately certified state and can thus be released safely to another state.