ELECTIVE III: SOFTWARE QUALITY ASSURANCE

SEMESTER: IV 2018-19 ONWARDS

SUBJECT CODE: 18MIT41E

UNIT I:

Introduction to software quality: what is software? – software errors, faults and failures- classification of causes of software errors – software quality – definition -software quality assurance – definition and objectives – software quality factors: the need for comprehensive software quality requirements – classification software requirements into software quality factors – product operation software quality factors – product revision software quality factors – product transition software quality factors – components of software quality assurance system – overview: the SQA system – an SQA architecture – pre project components – SQA standards, system certification and assessment components – organising for SQA – the human components.

UNIT II:

Development and quality plans: development plan and quality plan objectives – elements of development plan – elements of quality plan. Integrating quality objectives in the project life cycle: factors affecting intensity of quality assurance activities in the development process – verification, validation and qualification – a model for SQA defect removal effectiveness and cost. Reviews: review objectives – formal design reviews (DRs) - peer reviews – comparison of the team review methods – expert opinions.

UNIT III:

Assuring the quality of software maintenance components: the foundation of high quality – pre maintenance software quality components -maintenance software quality assurance tools. Assuring the quality of external participant contributions: types of external participants – risks and benefits of introducing external participants – assuring the quality of external participant contribution: objectives. CASE tools and their effect on software quality: CASE tool – contribution of CASE tool to software project quality, to software maintenance quality to improved project management.

UNIT IV:

Procedures and work instructions: the need – procedure manuals – work instruction manuals – preparation, implementation and updating. Supporting quality devices: templates – checklists. Staff training and certification: "3S" development team- the objectives – training and certification process – determining training and updating needs – defining positions requiring certification – planning the certification processes – delivery of training and certification programs.

UNIT V:

Software quality metrics: classification of software quality metrics – process metrics – product metrics – implementation of software quality metrics-quality management standards: ISO 9001 and ISO 9000-3 –certification according to ISO 9000 – 3- Capability maturity model – CMM and CMMI assessment methodology. SQA project process standards – IEEE software engineering standards: IEEE std 1012- verification and validation- IEEE std 1028 – reviews.

Text book:

Daniel Galin, “software quality assurance from theory to implementation”, pearson publication, 2009.

Reference books:

1. Alan C.Gillies , “software quality: theory and management”, international Thompson computer press,1997.
2. Mordechai Ben- Menachem “software quality: producing practical consistent software”, international Thompson computer press,1997.

UNIT I:

Introduction to software quality: what is software? – software errors, faults and failures- classification of causes of software errors – software quality – definition -software quality assurance – definition and objectives – software quality factors: the need for comprehensive software quality requirements – classification software requirements into software quality factors – product operation software quality factors – product revision software quality factors – product transition software quality factors – components of software quality assurance system – overview: the SQA system – an SQA architecture – pre project components – SQA standards, system certification and assessment components – organising for SQA – the human components.

Prepared by: Mrs. P.Sundari

Introduction to software quality

What is software?

An accumulation of programming language instructions and statements or development tool instructions that together form a program or software package. This program or software package is usually referred to as the “code”.

Software – IEEE definition

Software is:

Computer programs, procedures, and possibly associated documentation and data pertaining to the operation of a computer system.

The IEEE definition of software, which is almost identical to the ISO definition lists the following

Four components of software:

■ Computer programs (the “code”)

■ Procedures

■ Documentation

■ Data necessary for operating the software system.

All four components are needed in order to assure the quality of the software development process and the coming years of maintenance services for the following reasons:

■ Computer programs (the “code”) are needed because, obviously, they activate the computer to perform the required applications.

■ Procedures are required, to define the order and schedule in which the programs are performed, the method employed, and the person responsible for performing the activities that are necessary for applying the software.

■ Various types of documentation are needed for developers, users and maintenance personnel.

■ Data including parameters, codes and name lists that adapt the software to the needs of the specific user are necessary for operating the software. Another type of essential data is the standard test data, used to ascertain that no undesirable changes in the code or software data have occurred, and what kind of software malfunctioning can be expected.

Software errors, faults and failures

The origin of software failures lies in a **software error** made by a programmer.

An error can be a grammatical error in one or more of the code lines, or a logical error in carrying out one or more of the client's requirements.

However, not all software errors become **software faults**. In other words, in some cases, the software error can cause improper functioning of the software in general or in a specific application.

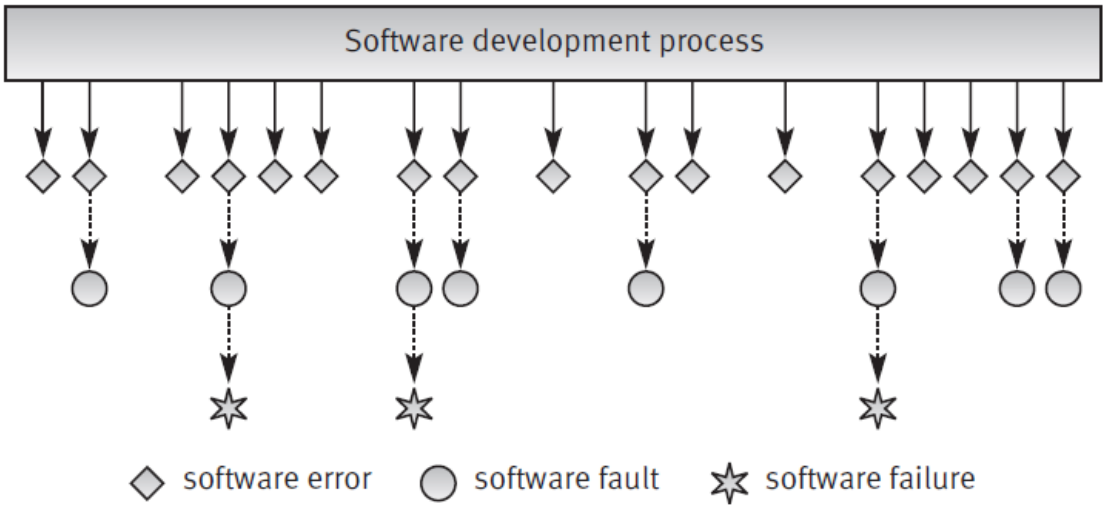
In many other cases, erroneous code lines will not affect the functionality of the software as a whole; in *a part* of these cases, the fault will be corrected or "neutralized" by subsequent code lines.

We are interested mainly in the software failures that disrupt our use of the software. This requires us to examine the relationship between software faults and **software failures**.

Do all software faults end with software failures?

Not necessarily: a software fault becomes a software failure only when it is "activated" – when the software user tries to apply the specific, faulty application.

In many situations, a software fault is never activated due to the user's lack of interest in the specific application or to the fact that the combination of conditions necessary to activate the fault never occurs.



SOFTWARE ERRORS, SOFTWARE FAULTS AND SOFTWARE FAILURES

Figure illustrates the relationships between software errors, faults and failures. In this figure, the development process yields 17 software errors, only eight of which become software faults. Of these faults, only three turnout to be software failures.

Importantly, developers and users have different views of the software product regarding its internal defects. While developers are interested in software errors and faults, their elimination, and the ways to prevent their generation, software users are worried about software failures.

Classification of the causes of software errors

A software error can be “code error”, a “procedure error”, a “documentation error”, or a “software data error”.

(1) Faulty definition of requirements

The faulty definition of requirements, usually prepared by the client, is one of the main causes of software errors. The most common errors of this type are:

■ Erroneous definition of requirements.

■ Absence of vital requirements.

■ Incomplete definition of requirements.

■ Inclusion of unnecessary requirements, functions that are not expected to be needed in the near future.

(2) Client–developer communication failures

Misunderstandings resulting from defective client–developer communication are additional causes for the errors that prevail in the early stages of the development process:

■ Misunderstanding of the client's instructions as stated in the requirement document.

■ Misunderstanding of the client's requirements changes presented to the developer in written form during the development period.

■ Misunderstanding of the client's requirements changes presented orally to the developer during the development period.

■ Misunderstanding of the client's responses to the design problems presented by the developer.

(3) Deliberate deviations from software requirements

The developer reuses software modules taken from an earlier project without sufficient analysis of the changes and adaptations needed to correctly fulfill all the new requirements.

■ Due to time or budget pressures, the developer decides to omit part of the required functions in an attempt to cope with these pressures.

■ Developer-initiated, unapproved improvements to the software, introduced without the client's approval, frequently disregard requirements that seem minor to the developer. Such "minor" changes may, eventually, cause software errors.

(4) Logical design errors

■ Definitions that represent software requirements by means of erroneous algorithms.

■ Process definitions that contain sequencing errors.

■ Erroneous definition of boundary conditions.

■ Omission of required software system states.

■ Omission of definitions concerning reactions to illegal operation of the software system.

(5) Coding errors

A broad range of reasons cause programmers to make coding errors. These include misunderstanding the design documentation, linguistic errors in the programming languages, errors in the application of CASE and other development tools, errors in data selection, and so forth.

(6) Non-compliance with documentation and coding instructions

■ Team members who need to coordinate their own codes with code modules developed by "non-complying" team members can be expected to encounter more than the usual number of difficulties when trying to understand the software developed by the other team members.

■ Individuals replacing the "non-complying" team member (who has retired or been promoted) will find it difficult to fully understand his or her work.

■ The design review team will find it more difficult to review a design prepared by a non-complying team.

(7) Shortcomings of the testing process

■ Incomplete test plans leave untreated portions of the software or the application functions and states of the system.

■ Failures to document and report detected errors and faults.

■ Failure to promptly correct detected software faults as a result of inappropriate indications of the reasons for the fault.

■ Incomplete correction of detected errors due to negligence or time pressures.

(8) Procedure errors

Procedures direct the user with respect to the activities required at each step of the process.

(9) Documentation errors

Typical errors of this type are:

■ Omission of software functions.

■ Errors in the explanations and instructions given to users, resulting in “dead ends” or incorrect applications.

■ Listing of non-existing software functions, that is, functions planned in the early stages of development but later dropped, and functions that were active in previous versions of the software but cancelled in the current version.

Software quality – IEEE definition

Software quality is:

1. The degree to which a system, component, or process meets specified requirements.

2. The degree to which a system, component, or process meets customer or user needs or expectations.

- "Quality means conformance to requirements"
- Quality consists of those product features which meet the needs of customers and thereby provide product satisfaction.

Quality consists of freedom from deficiencies.

Software quality – Pressman's definition

Software quality is defined as:

Conformance to explicitly stated functional and performance requirements, explicitly documented development standards, and implicit characteristics that are expected of all professionally developed software.

Software quality assurance – definition and objectives

Software quality assurance – The IEEE definition

Software quality assurance is:

1. A planned and systematic pattern of all actions necessary to provide adequate confidence that an item or product conforms to established technical requirements.

2. A set of activities designed to evaluate the process by which the products are developed or manufactured. Contrast with quality control.

SQA – expanded definition

Software quality assurance is:

A systematic, planned set of actions necessary to provide adequate confidence that the software development process or the maintenance process of a software system product conforms to established functional

technical requirements as well as with the managerial requirements of keeping the schedule and operating within the budgetary confines.

The objectives of SQA activities

Software development (process-oriented):

1. Assuring an acceptable level of confidence that the software will conform to functional technical requirements.

2. Assuring an acceptable level of confidence that the software will conform to managerial scheduling and budgetary requirements.

3. Initiating and managing of activities for the improvement and greater efficiency of software development and SQA activities. This means improving the prospects that the functional and managerial requirements will be achieved while reducing the costs of carrying out the software development and SQA activities.

Software maintenance (product-oriented):

1. Assuring with an acceptable level of confidence that the software maintenance activities will conform to the functional technical requirements.

2. Assuring with an acceptable level of confidence that the software maintenance activities will conform to managerial scheduling and budgetary requirements.

3. Initiating and managing activities to improve and increase the efficiency of software maintenance and SQA activities. This involves improving the prospects of achieving functional and managerial requirements while reducing costs.

Software quality factors

The need for comprehensive software quality requirements

There is a need for a comprehensive definition of requirements that will cover all attributes of software and aspects of the use of software, including usability aspects, reusability aspects, maintainability aspects, and so forth in order to assure the full satisfaction of the users.

Classifications of software requirements into software quality factors

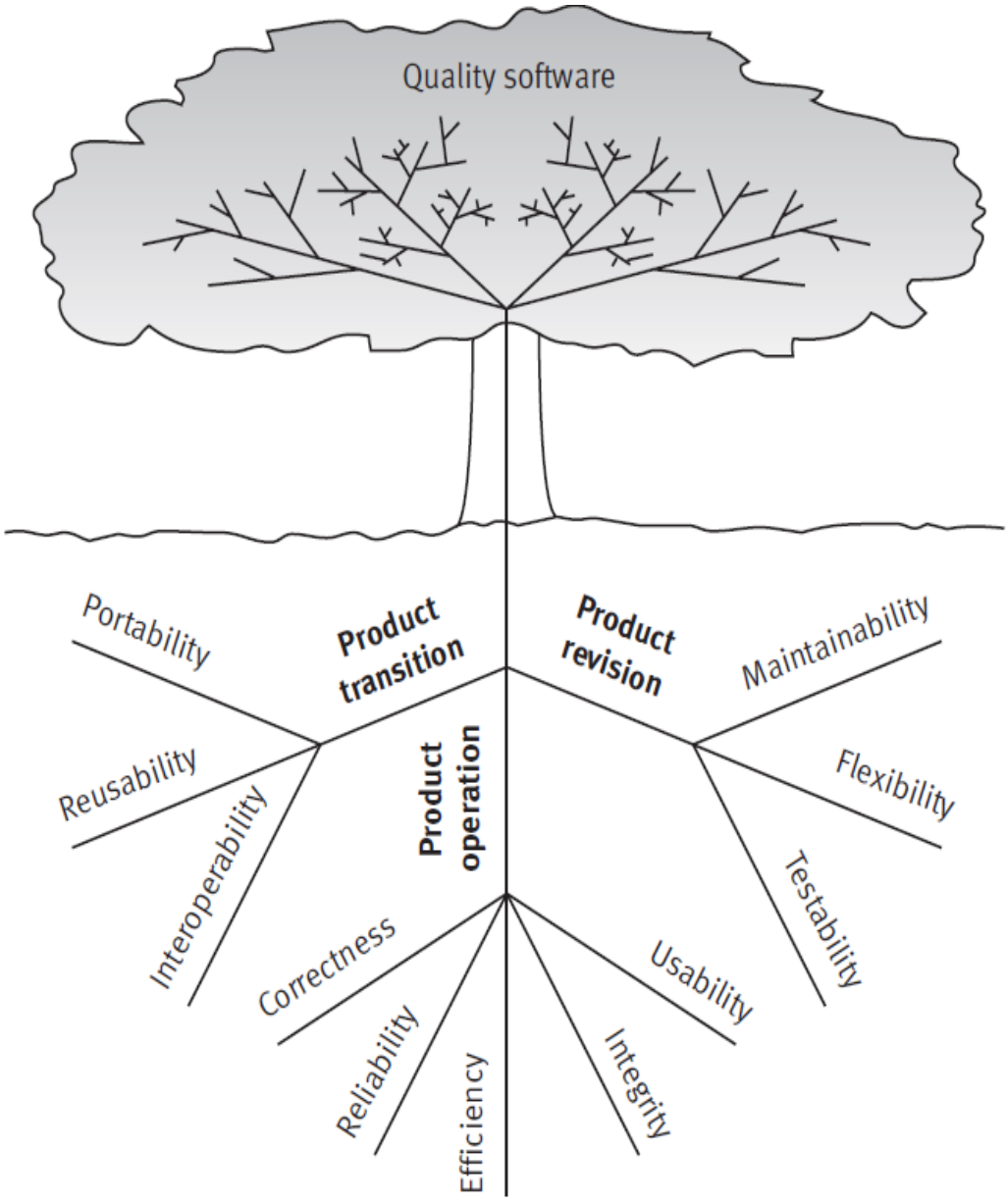
McCall's factor model

McCall's factor model classifies all software requirements into 11 software quality factors. The 11 factors are grouped into three categories – product operation, product revision and product transition – as follows:

- **Product operation factors:** Correctness, Reliability, Efficiency, Integrity, Usability.

- **Product revision factors:** Maintainability, Flexibility, Testability.

- **Product transition factors:** Portability, Reusability, Interoperability.



McCall's factor model tree

Product operation software quality factors

Correctness

- The required accuracy of those outputs.
- The completeness of the output information.
- The up-to-dateness of the information.
- The availability of the information.
- The standards for coding and documenting the software system.

Reliability

Reliability requirements deal with failures to provide service. They determine the maximum allowed software system failure rate, and can refer to the entire system or to one or more of its separate functions.

Efficiency

Efficiency requirements deal with the hardware resources needed to perform all the functions of the software system in conformance to all other requirements.

Integrity

Integrity requirements deal with the software system security, that is, requirements to prevent access to unauthorized persons, to distinguish between the majority of personnel allowed to see the information (“read permit”) and limited group who will be allowed to add and change data (“write permit”) and so forth.

Usability

Usability requirements deal with the scope of staff resources needed to train a new employee and to operate the software system.

Product revision software quality factors

Maintainability

Typical maintainability requirements:

(a) The size of a software module will not exceed 30 statements.

(b) The programming will adhere to the company coding standards and guidelines.

Flexibility

The capabilities and efforts required to support adaptive maintenance activities are covered by the flexibility requirements. These include the resources (i.e. in man-days) required to adapt a software package to a variety of customers of the same trade, of various extents of activities, of different ranges of products and so on. This factor's requirements also support perfective maintenance activities, such as changes and additions to the software in order to improve its service and to adapt it to changes in the firm's technical or commercial environment.

Testability

Testability requirements deal with the testing of an information system as well as with its operation.

Testability requirements related to software operation include automatic diagnostics performed by the software system prior to starting the system, to find out whether all components of the software system are in working order and to obtain a report about the detected faults.

Product transition software quality factors

Portability

Portability requirements tend to the adaptation of a software system to other environments consisting of different hardware, different operating systems, and so forth. These requirements make it possible to continue using the same basic software in diverse situations or to use it simultaneously in diverse hardware and operating systems situations.

Reusability

Reusability requirements deal with the use of software modules originally designed for one project in a new software project currently being developed. They may also enable future projects to make use of a given module or a group of modules of the currently developed software. The reuse of software is expected to save development resources, shorten the development period, and provide higher quality modules.

Interoperability

Interoperability requirements focus on creating interfaces with other software systems or with other equipment firmware.

Interoperability requirements can specify the name(s) of the software or firmware for which interface is required.

The components of the software quality assurance system – overview

The SQA system – an SQA architecture

SQA system components can be classified into six classes:

■ **Pre-project components**. To assure that (a) the project commitments have been adequately defined considering the resources required, the schedule and budget; and (b) the development and quality plans have been correctly determined.

■ **Components of project life cycle activities assessment**. The project life cycle is composed of two stages: the development life cycle stage and the operation–maintenance stage. The development life cycle stage components detect design and programming errors. Its components are divided into the following four sub-classes:

– Reviews

– Expert opinions

– Software testing.

The SQA components used during the operation–maintenance phase include specialized maintenance components as well as development life cycle components, which are applied mainly for functionality improving maintenance tasks. An additional sub-class of SQA project life cycle components deals with assuring the quality of project parts performed by subcontractors and other external participants during project development and maintenance.

■ **Components of infrastructure error prevention and improvement**. The main objectives of these components, which are applied throughout the entire organization, are to eliminate or at least reduce the rate of errors, based on the organization's accumulated SQA experience.

■ **Components of software quality management**. This class of components is geared toward several goals, the major ones being the control of development and maintenance activities and the introduction of early managerial support actions that mainly prevent or minimize schedule and budget failures and their outcomes.

■ **Components of standardization, certification, and SQA system assessment**.

These components implement international professional and managerial standards within the organization. The main objectives of this class are

(a) utilization of international professional knowledge,

(b) improvement of coordination of the organizational quality systems with other organizations, and

(c) assessment of the achievements of quality systems according to a common scale.

The various standards may be classified into two main groups:

(a) quality management standards, and

(b) project process standards.

■ **Organizing for SQA – the human components**. The SQA organizational base includes managers, testing personnel, the SQA unit and practitioners interested in software quality (SQA trustees, SQA committee members and SQA forum members). All these *actors* contribute to software quality; their main objectives are to initiate and support the implementation of SQA components, detect deviations from SQA procedures and methodology, and suggest improvements.

SQA system component classes

Pre-project quality components

Project life cycle quality components

Infrastructure error preventive and improvement components

Software quality management components

Standardization, certification and SQA assessment components

Organizing for SQA – the human components

SQA Architecture

Pre-project components

Contract review

Accordingly, contract review activities must include a detailed examination of (a) the project proposal draft and (b) the contract drafts. Specifically, contract review activities include:

- Clarification of the customer's requirements
- Review of the project's schedule and resource requirement estimates
- Evaluation of the professional staff's capacity to carry out the proposed project
- Evaluation of the customer's capacity to fulfill his obligations
- Evaluation of development risks.

Development and quality plans

Once a software development contract has been signed or a commitment made to undertake an internal project for the benefit of another department of the organization, a plan is prepared of the project ("development plan") and its integrated quality assurance activities ("quality plan").

The main issues treated in the project development plan are:

- Schedules
- Required manpower and hardware resources
- Risk evaluations
- Organizational issues: team members, subcontractors and partnerships
- Project methodology, development tools, etc.
- Software reuse plans.

The main issues treated in the project's quality plan are:

- Quality goals, expressed in the appropriate measurable terms
- Criteria for starting and ending each project stage
- Lists of reviews, tests, and other scheduled verification and validation activities.

SQA standards, system certification, and assessment components

External tools offer another avenue for achieving the goals of software quality assurance. Specifically, the main objectives of this class of components are:

(1) Utilization of international professional knowledge.

(2) Improvement of coordination with other organizations' quality systems.

(3) Objective professional evaluation and measurement of the achievements of the organization's quality systems.

The standards available may be classified into two main sub-classes: quality management standards and project process standards. Either or both of the two sub-classes can be required by the customer and stipulated in the accompanying contractual agreements.

Quality management standards

These standards focus on *what* is required and leave the decision about *how* to achieve it to the organization.

The most familiar examples of this type of standard are:

- SEI CMM assessment standard
- ISO 9001 and ISO 9000-3 standards.

Project process standards

Project process standards are professional standards that provide methodological guidelines (dealing with the question of "how") for the development team. Well-known examples of this type of standards are:

- IEEE 1012 standard
- ISO/IEC 12207 standard.

Organizing for SQA – the human components

Management's role in SQA

The responsibilities of top management (through the executive in charge of software quality), departmental management and project management include the following:

- Definition of the quality policy
- Effective follow-up of quality policy implementation
- Allocation of sufficient resources to implement quality policy
- Assignment of adequate staff
- Follow-up of compliance of quality assurance procedures
- Solutions of schedule, budget and customer relations difficulties.

The SQA unit

This unit and software testers are the only parts of the SQA organizational base that devote themselves full-time to SQA matters. All other segments of the SQA organizational base (managerial and professional staff) contribute only some of their time to software quality issues. Thus, the SQA unit's task is to serve as the main moving force, initiator, and coordinator of the SQA system and its application. This task can be broken down into a number of primary roles:

■ Preparation of annual quality programs

■ Consultation with in-house staff and outside experts on software quality issues

■ Conduct of internal quality assurance audits

■ Leadership of quality assurance various committees

■ Support of existing quality assurance infrastructure components and their updates, and development of new components.

SQA trustees, committees and forums

SQA trustees are members of development and maintenance teams who have a special interest in software quality and are prepared to devote part of their time to these issues. Their contributions include:

■ Solving team or unit local quality problems

■ Detecting deviations from quality procedures and instructions

■ Initiating improvements in SQA components

■ Reporting to the SQA unit about quality issues in their team or unit.

SQA committee members are members of various software development and maintenance units, and are usually appointed for term or *ad hoc* service. The main issues dealt with by the committees are:

■ Solution of software quality problems.

■ Analysis of problem and failure records as well as other records, followed by initiation of corrective and preventive actions when appropriate.

■ Initiation and development of new procedures and instructions; updating existing materials.

- Initiation and development of new SQA components and improvement of existing components.

Organizing for SQA – the human components

SQA forums are composed of professionals and practitioners who meet and/or maintain an Internet site on a voluntary basis for discussion of quality issues pertaining to development and maintenance processes. They share their experiences and difficulties as well as try to initiate improvements in the software process. The forums can therefore be considered as important sources of information and SQA initiatives.