Lecture Notes for Lecture 13 of CS 5500 (Foundations of Software Engineering) for the Spring 2021 session at the Northeastern University San Francisco Bay Area Campuses.

Software Quality Assurance

Philip Gust,
Clinical Instructor
Department of Computer Science

Information and examples in this lecture are based on recommended readings.

http://www.ccis.northeastern.edu/home/pgust/classes/cs5500/2021/Spring/index.html

Software Quality Assurance

Review of Lecture 12

- In lecture 12, we continued our discussion about managing and tracking in software development
- We know managing and tracking software development enables teams to assess progress against their own goals and estimates, and to make adjustments as soon as possible to stay on track.
- As we also know that measurement is an important aspect of estimation; managing and tracking software development provides information to improve future estimates.
- In this lecture, we explored managing and tracking development processes. This is a complex topic because so many aspects of development are governed by processes that must be coordinated.
- We also discussed tools for managing and tracking software development processes.

Software Quality Assurance

Introduction

- In this lecture, we introduce the concepts of quality and software quality assurance (SQA), including its processes and characteristics. We will also cover approaches to SQA, and metrics, and statistics.
- SQA and software testing is a recognized field of software engineering that merits its own courses. This introduction is meant to give developers an idea of its goals, strategies, and practice.
- We will conclude the discussion next week as we cover software testing strategies, and the practice of software testing.

Software Quality Assurance

What is quality?

- In his book *Managing Quality: The Strategic and Competitive Edge*, David Garvin suggests 5 major approaches to quality:.
  - The **transcendental view** argues that quality is something you immediately recognize, but cannot explicitly define.
  - The **user view** sees quality in terms of an end user's specific goals. If a product meets those goals, it exhibits quality.
  - The **manufacturer's view** defines quality in terms of the specification of the product. If the product conforms, it exhibits quality.
  - The **product view** suggests that quality can be tied to inherent characteristics (e.g., functions and features) of a product.
  - The **value-based view** measures quality based on how much a customer is willing to pay for a product.

Software Quality Assurance

What is quality?

- *Quality of design* refers to characteristics that designers specify for a product. It encompasses the degree to which the design meets the functions and features specified in the requirements model.
- *Quality of conformance* focuses on the degree to which the implementation follows the design, and the resulting system meets its requirements and performance goals.

Software Quality Assurance

What is quality?

- Are quality of design and quality of conformance the only issues that software engineers must consider?
- According to software quality expert Robert Glass, a more intuitive relationship is that user satisfaction is a combination of three factors:
  - compliant product
  - good quality
  - delivery within budget and schedule

- The bottom line is that quality is important, but if the user isn't satisfied, nothing else really matters.

Software Quality Assurance

What is quality?

- Software engineering pioneer Tom DeMarco reinforces this view when he states: “A product’s quality is a function of how much it changes the world for the better.”
- This view of quality contends that if a software product provides substantial benefit to its end users, they may be willing to tolerate occasional reliability or performance problems.

Software Quality Assurance

What is software quality?

- High-quality software is an important goal, but how do we define software quality?
- In the most general sense, software quality can be defined as:

  An effective software process applied in a manner to create a useful product that provides measurable value for those who produce it and those who use it.

- This definition could be modified, extended, and debated, but for this discussion, the definition emphasizes three important points.

Software Quality Assurance

What is software quality?

1. An *effective software process* establishes the infrastructure that supports any effort at building a high-quality software product.
   - The management aspects of process create checks and balances that help avoid project chaos—a key contributor to poor quality.
   - Software engineering practices allow the developer to analyze the problem and design a solid solution— both critical to building high-quality software.
   - Umbrella activities such as change management and technical reviews have as much to do with quality as any other part of software engineering practice.

Software Quality Assurance

What is software quality?

2. A *useful product* delivers the content, functions, and features that the end user desires, but just as important, it delivers these assets in a reliable, error-free way.
   - A useful product always satisfies those requirements that have been explicitly stated by stakeholders.
   - In addition, it satisfies a set of implicit requirements, such as ease of use, that users expect of all high-quality software.

Software Quality Assurance

What is software quality?

3. By *adding value* for both the producer and user, high-quality software provides benefits for the software organization and the end-user community.
   - The software organization gains added value because high-quality software requires less maintenance effort, fewer bug fixes, and reduced customer support.
   - The user community gains added value because the application provides a useful capability in a way that expedites some business process.
   - The result is:
     - greater product revenue,
     - better profitability when an application supports a business process,
     - improved availability of information that is crucial for the business.

Software Quality Assurance

Garvin's Quality Dimensions

- Several ways have been proposed to characterize software quality.
- David Garvin suggests quality should be considered from a multi-dimensional viewpoint.
- Garvin's *eight dimensions of quality* were not developed specifically for software, but they can be applied when software quality is considered.

Software Quality Assurance

Garvin's Quality Dimensions

1. **Performance Quality**. Does the software deliver all content, functions, and features specified by the requirements model in a way that provides value to the end user?
2. **Feature quality**. Does the software provide features that surprise and delight first-time end users?
3. **Reliability**. Does the software deliver all of the features and capability without failure? Is it available when it is needed? Does it deliver functionality that is error free?
4. **Conformance**. Does the software conform to local and external software standards that are relevant to the application? Does it conform to de facto design and coding conventions?

Software Quality Assurance

Garvin's Quality Dimensions

5. **Durability**. Can the software be maintained without inadvertent generation of unintended side effects? Will changes cause the error rate or reliability to degrade with time?
6. **Serviceability**. Can the software be maintained in an acceptably short time period? Do staff have information they need to make changes?
7. **Aesthetics**. Aesthetic entities have elegance, a flow, and a "presence" that is hard to quantify. Aesthetic software also has these characteristics.
8. **Perception**. Prejudices can influence perception of quality. If a new software product is from a vendor with poor quality your guard is raised and your perception of the new product might be influenced negatively. Similarly, if a vendor has an excellent reputation, you may perceive quality, even when it does not really exist.

Software Quality Assurance

Garvin's Quality Dimensions

- Garvin's quality dimensions provide a "soft" look at software quality. Many, but not all these dimensions can only be considered subjectively.
- For this reason, you also need a set of "hard" quality factors that can be categorized in two broad groups:
  1. factors that can be directly measured (e.g., defects uncovered during testing) and
  2. factors that can be measured only indirectly (e.g., usability or maintainability).
- In each case, measurement must occur. You should compare the software to datum and arrive at an indication of quality.

Software Quality Assurance

ISO 9126 Quality Factors

- The International Standards Organization *ISO 9126* standard was developed to identify the key quality attributes for computer software. The standard identifies 6 key quality factors.
- The ISO 9126 factors do not necessarily lend themselves to direct measurement. However, they provide a basis for indirect measures and a checklist for assessing the quality of a system.
- The ISO 9126 standard is divided into 4 parts:
  - quality model
  - external metrics
  - internal metrics
  - quality in use metrics
- *Note: ISO 9126 was replaced by ISO 25010:2011, which added "security" and "compatibility" as factors.*

Software Quality Assurance

ISO 9126 Quality Factors

Software Quality Assurance

ISO 9126 Quality Factors

- **Functionality**. The degree to which the software satisfies stated needs, indicated by suitability, accuracy, interoperability, compliance, and security sub-attributes.
- **Reliability**. The amount of time that the software is available for use as indicated by maturity, fault tolerance, and recoverability sub-attributes.
- **Usability**. The degree to which the software is easy to use as indicated by understandability, learnability, and operability sub-attributes.
- **Efficiency**. The degree to which the software makes optimal use of system resources as indicated by time and resource behavior sub-attributes..
- **Maintainability**. The ease with which repair may be made to the software as indicated by analyzability, changeability, stability, and testability sub-attributes.
- **Portability**. The ease with which the software can be transposed from one environment to another as indicated by adaptability, installability, conformance, and replaceability sub-attributes.

Software Quality Assurance

Achieving software quality

- Software quality doesn't just appear. It is the result of good project management and solid software engineering practice.
- Management and practice are applied within the context of four broad activities that help a software team achieve high software quality:
  - software engineering methods,
  - project management techniques,
  - quality control actions,
  - software quality assurance.

Software Quality Assurance

Software engineering methods

- If you expect to build high-quality software, you must understand the problem to be solved.
- You must also create a design that conforms to the problem while incorporating characteristics that lead to software that exhibits the required quality dimensions and the factors we just discussed.

Software Quality Assurance

Project management techniques

- The impact of poor management decisions on software quality are clear: software quality will be affected in a positive way if:
  - estimations are used to verify that delivery dates are achievable,
  - schedule dependencies are understood, and the team resists the temptation to use shortcuts,
  - risk planning is conducted so problems do not breed chaos.
- In addition, the project plan should include explicit techniques for quality and change management.

Software Quality Assurance

Quality control

- Quality control encompasses a set of software engineering actions that help to ensure that each work product meets its quality goals.
  - Models are reviewed to ensure that they are complete and consistent.
  - Code may be inspected in order to uncover and correct errors before testing commences.
  - A series of testing steps is applied to uncover errors in processing logic, data manipulation, and interface communication.
  - A combination of measurement and feedback allows a software team to tune the process when any of these work products fail to meet quality goals.

Software Quality Assurance

Quality assurance

- *Quality assurance* establishes the infrastructure that supports solid software engineering methods, rational project management, and quality control actions—all pivotal to high-quality software.
- In addition, quality assurance consists of a set of auditing and reporting functions that assess the effectiveness and completeness of quality control actions.
- Quality assurance provides management and technical staff with data to be informed about product quality, gain insight and build confidence that actions to achieve product quality are working.
- If data from quality assurance identifies problems, problems must be addressed and the necessary resources must be applied to resolve quality issues.

Software Quality Assurance

The cost of quality

- The cost of quality includes all costs incurred in the pursuit of quality or in performing quality-related activities and the downstream costs of lack of quality.
- To understand these costs, metrics should be collected to
  - provide a baseline for the current cost of quality,
  - identify opportunities for reducing these costs, and
  - provide a normalized basis of comparison.
- The cost of quality can be divided into the costs associated with prevention, appraisal, and failure.

Software Quality Assurance

The cost of quality

- **Prevention costs** include:
  - the cost of management activities required to plan and coordinate all quality control and quality assurance activities,
  - the cost of added technical activities to develop complete requirements and design models,
  - test planning costs,
  - the cost of all training associated with these activities.

Software Quality Assurance

The cost of quality

- **Appraisal costs** include activities to gain insight into product condition the “first time through” each process.
- Examples of appraisal costs include:
  - the cost of conducting technical reviews for software engineering work products,
  - the cost of data collection and metrics evaluation,
  - the cost of testing and debugging.

Software Quality Assurance

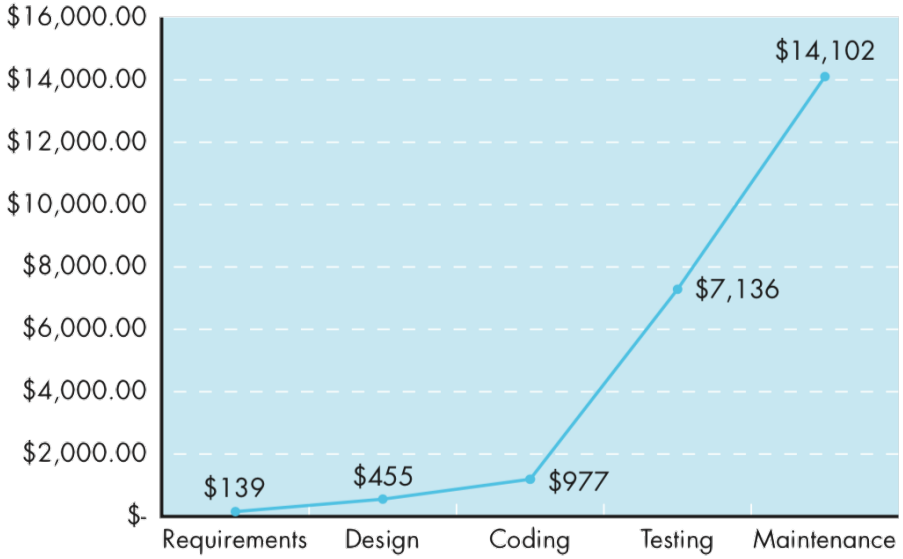
The cost of quality

- **Failure costs** are those that would disappear if no errors appeared before shipping a product to customers. These costs may be subdivided into:
  - *Internal failure costs:* incurred when errors in a product are found prior to shipment, including
    - cost to perform rework (repair) to correct an error,
    - cost when rework that generates side effects must be mitigated,
    - costs to collect quality metrics to assess the modes of failure.
  - *External failure costs*: incurred with defects found after the product has been shipped to the customer, including
    - complaint resolution,
    - product return and replacement,
    - help line support,
    - labor costs associated with warranty work.

Software Quality Assurance

The cost of quality

- The relative costs to find and repair an error or defect increase dramatically as we go from prevention to detection to internal failure to external failure costs
- The industry average cost to correct a defect during code generation is around $977 per error.
- The Industry average cost to correct the same error during system testing is $7,136 per error.
- The cost goes to $14,102 per error in maintenance.



Software Quality Assurance

The cost of quality

- For a moderately large software project, there may be as many as 200 significant defects. If all 200 were found during coding, the cost would be (200 x $977) = $195,400.
- If none were found during coding and system testing only finds 50 (25%) of the defects, the cost of finding and fixing them would be (50 x $7,137) = $356,800.
- That leaves the other 150 to find and fix in maintenance, at a cost of (150 x $14,102) = $2,115,300.
- The difference between the two scenarios is $195,400 versus ($356,800 + $2,115,300) = $2,472,100.

Software Quality Assurance

The cost of quality

- Even if your organization has costs half of industry average, savings associated with early quality control and assurance activities, conducted during requirements analysis and design, are significant.
- Worse still, most software organizations have no idea of their costs because they do not track or analyze that information.

Software Quality Assurance

Software Quality Assurance (SQA)

- Now that we have a good idea about quality and some of the issues, we will next turn our attention to software quality assurance (SQA).
- We will learn about what SQA means and techniques to achieve it as part of the quality control process.
- Our focus will be on the management issues and the process-specific activities that enable a software organization to ensure that it does “the right things at the right time in the right way.”

Software Quality Assurance

What is SQA?

- Software quality assurance (SQA) encompasses the following elements of software development.
  - a well-defined SQA process,
  - specific quality assurance and quality control tasks (including technical reviews and a multi-tiered testing strategy),
  - effective software engineering practice (methods and tools),
  - control of all software work products and the changes made to them,
  - a procedure to ensure compliance with software development standards (when applicable),
  - measurement and reporting mechanisms.

Software Quality Assurance

Background on QA

- Prior to the 20th century, quality control was the responsibility of the craftsperson who built a product.
- As mass production techniques became commonplace, quality control became an activity performed by people other than the ones who built the product.
- The first formal quality assurance and control function was introduced at Bell Labs in 1916 and spread rapidly throughout the manufacturing world.
- During the 1940s, more formal approaches to quality control were suggested. These relied on measurement and continuous process improvement as key elements of quality management

Software Quality Assurance

Advent of QA in Software

- The history of quality assurance in software development parallels the history of quality in hardware manufacturing.
- During the early days of computing (1950s and 1960s), quality was the sole responsibility of the programmer.
- Standards for quality assurance for software were introduced in military contract software development during the 1970s and have spread rapidly into software development in the commercial world.

Software Quality Assurance

Advent of QA in Software

- Software quality assurance is a planned and systematic pattern of actions that are required to ensure high quality in software.
- Many different constituencies have software quality assurance responsibility, including:
  - software engineers,
  - project managers,
  - customers,
  - salespeople,
  - individuals who serve within an SQA group.

Software Quality Assurance

Role of the SQA group

- The SQA group serves as the customer's in-house representative. The people who perform SQA must look at the software from the customer's point of view.
- The SQA group attempts to answer these and other questions to ensure software quality is maintained.
  - Does the software adequately meet established quality factors?
  - Have software engineering practices been conducted according to established standards?
  - Have technical disciplines properly performed their roles as part of the SQA activity?

Software Quality Assurance

Role of the SQA group

- Software quality assurance encompasses a broad range of concerns and activities that focus on the management of software quality.
  - **Standards**. SQA adopts and puts SQA standards into practice from the IEEE, ISO, and other standards organizations.
  - **Reviews and audits**. SQA personnel perform audits to ensure that quality guidelines are being followed for software engineering work.
  - **Testing**. SQA ensures that testing is properly planned and efficiently conducted so it has the highest likelihood of achieving its goal.
  - **Error/defect collection and analysis**. SQA collects and analyzes data to understand how errors are introduced and how to eliminate them.
  - **Change management**. SQA ensures that adequate change management practices have been instituted.
  - **Education**. The SQA organization takes the lead in software process improvement and is a key proponent of educational programs.

Software Quality Assurance

SQA processes and approaches

- SQA processes and approaches that work in one software environment may not work as well in another.
- Even within a company that adopts a consistent approach to software engineering, different software products may have different levels of quality.
- The solution is to understand the specific quality requirements for a software product, and then select the process, and specific SQA actions and tasks to meet those requirements.

Software Quality Assurance

SQA group activities

- The charter of the SQA group is to assist the software team in achieving a high-quality end product.
- The Software Engineering Institute (SEI) recommends a set of SQA activities that address quality assurance planning, oversight, record keeping, analysis, and reporting.
- These activities are performed (or facilitated) by an independent SQA group. Here are just a few.

Software Quality Assurance

SQA group activities

- Prepares an SQA plan for a project.
- Participates in development of project's software process description.
- Reviews software engineering activities to verify compliance with the defined software process.
- Audits designated software work products to verify compliance with those defined as part of the software process.
- Ensures that deviations in software work and work products are documented and handled according to a documented procedure.
- Records any noncompliance and reports to senior management.
- Coordinates the control and management of change.
- Helps to collect and analyze software metrics.

Software Quality Assurance

SQA group activities

- The SQA activities achieve a set of pragmatic goals:
  - **Requirements quality**. The correctness, completeness, and consistency of the requirements model strongly influences the quality of all work products that follow. SQA must ensure that the software team has properly reviewed the requirements model to achieve a high level of quality.
  - **Design quality**. Every element of the design model should be assessed by the software team to ensure that it exhibits high quality and the design conforms to requirements. SQA looks for attributes that are indicators of quality.
  - **Code quality**. Source code and related work products (e.g., other descriptive information) must conform to local coding standards and have characteristics that facilitate maintainability. SQA should isolate those attributes that allow a reasonable analysis of the quality of code.
  - **Quality control effectiveness**. A software team should apply limited resources in a way that has the highest likelihood of achieving a high-quality

Software Quality Assurance

Statistical quality assurance

- Statistical quality assurance reflects a growing trend throughout the industry to become more quantitative about quality. For software, statistical quality assurance implies the following steps:
  1. Information about software errors and defects is collected and categorized.
  2. An attempt is made to trace each error and defect to its underlying cause (e.g., nonconformance to specifications, design error, violation of standards, poor communication with the customer).
  3. Using the *Pareto principle* (80 percent of the defects can be traced to 20 percent of all possible causes), isolate the 20 percent (the vital few).
  4. Once the vital few causes have been identified, move to correct the problems that have caused the errors and defects.
- This contributes to creating an adaptive process, where changes are made to improve elements of the process that introduce error.

Software Quality Assurance

Generic example of statistical SQA

- To illustrate the use of statistical methods for software engineering work, assume that a software engineering organization collects information on errors and defects for a period of one year.
- Some of the errors are uncovered as software is being developed. Other defects are encountered after the software has been released to its end users.
- Although hundreds of different problems are uncovered, all can be tracked to one (or more) causes.

Software Quality Assurance

Generic example of statistical SQA

- Incomplete or erroneous specifications (**IES**).
- Misinterpretation of customer communication (**MCC**).
- Intentional deviation from specifications (**IDS**).
- Violation of programming standards (**VPS**).
- Error in data representation (**EDR**).
- Inconsistent component interface (**ICI**).
- Error in design logic (**EDL**).
- Incomplete or erroneous testing (**IET**).
- Inaccurate or incomplete documentation (**IID**).
- Error in programming language translation of design (**PLT**).
- Ambiguous or inconsistent human/computer interface (**HCI**).
- Miscellaneous (**MIS**).

Software Quality Assurance

Generic example of statistical SQA

- To apply statistical SQA, we build the following table with our data:

	Total		Serious		Moderate		Minor	
Error	**No.**	**%**	**No.**	**%**	**No.**	**%**	**No.**	**%**
IES	205	22%	34	27%	68	18%	103	24%
MCC	156	17%	12	9%	68	18%	76	17%
IDS	48	5%	1	1%	24	6%	23	5%
VPS	25	3%	0	0%	15	4%	10	2%
EDR	130	14%	26	20%	68	18%	36	8%
ICI	58	6%	9	7%	18	5%	31	7%
EDL	45	5%	14	11%	12	3%	19	4%
IET	95	10%	12	9%	35	9%	48	11%
IID	36	4%	2	2%	20	5%	14	3%
PLT	60	6%	15	12%	19	5%	26	6%
HCI	28	3%	3	2%	17	4%	8	2%
MIS	56	6%	0	0%	15	4%	41	9%
Totals	942	100%	128	100%	379	100%	435	100%

Software Quality Assurance

Generic example of statistical SQA

- The table indicates that these are the vital few causes that account for 53 percent of all errors.
  - Incomplete or erroneous specifications (**IES**) – 22%.
  - Misinterpretation of customer communication (**MCC**) – 17%.
  - Error in data representation (**EDR**) – 14%.
- Note, however, that these would be selected as the vital few causes if only serious errors are considered.
  - Incomplete or erroneous specifications (**IES**) – 27%.
  - Error in data representation (**EDR**) – 26%.
  - Error in programming language translation of design (**PLT**) – 12%.
  - Error in design logic (**EDL**) – 11%.

Software Quality Assurance

Generic example of statistical SQA

- Once the *vital few* causes are determined, the software engineering organization can begin corrective action.
- For example, to correct **MCC** (misinterpretation of customer communication), you might implement requirements techniques to improve quality of customer communication and specifications.
- To improve **EDR** (error in data representation), you might acquire tools for data modeling and perform more stringent data design reviews.
- Note that corrective action focuses primarily on the vital few. As the vital few causes are corrected, new candidates rise to the top.

Software Quality Assurance

Statistical SQA summary

- Statistical SQA techniques have shown to provide substantial quality improvement. In some cases, software organizations have had a 50 percent reduction per year in defects
- The application of statistical SQA and Pareto principle (vital few) can be summarized in a single sentence:
  - *Spend your time focusing on things that really matter, but first be sure that you understand what really matters!*

Software Quality Assurance

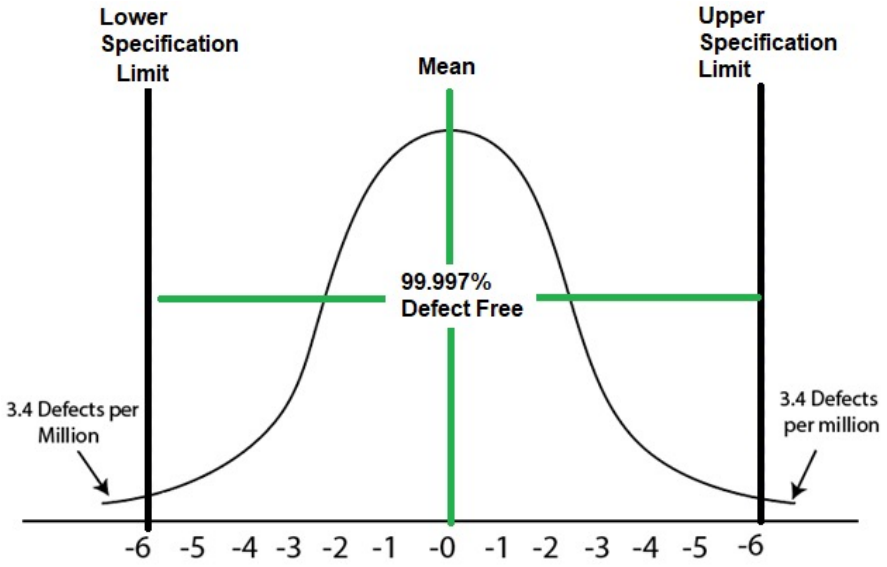
Six Sigma for software engineering

- *Six Sigma* is the most widely used strategy for statistical quality assurance in industry today. It was originally popularized by Motorola in the 1980s.
- The Six Sigma strategy is a rigorous and disciplined methodology that uses data and statistical analysis to measure and improve a company's operational performance.
- It works by identifying and eliminating defects in manufacturing and service-related processes.

Software Quality Assurance

Six Sigma for software engineering

- The term Six Sigma is derived from six standard deviations—3.4 instances (defects) per million occurrences—implying an extremely high-quality standard.



Six Sigma Curve

Software Quality Assurance

Six Sigma for software engineering

- The Six Sigma methodology defines these core steps:
  - **Define** customer requirements and deliverables and project goals via well-defined methods of customer communication.
  - **Measure** the existing process and it's output to determine current quality performance (collect defect metrics).
  - **Analyze** defect metrics and determine the vital few causes.
  - **Improve** the process based on the results of the analysis.
  - **Control** the future state of the process to correct deviations before they result in defects.

Software Quality Assurance

Six Sigma for software engineering

- If an existing software process is in place, but improvement is required, Six Sigma suggests two additional steps:
  - Improve the process by eliminating the root causes of defects.
  - Control the process to ensure that future work does not reintroduce the causes of defects.

Software Quality Assurance

Six Sigma for software engineering

- These core and additional steps are sometimes referred to as the DMAIC (Define, Measure, Analyze, Improve, and Control) method.
- If an organization is developing a software process rather than improving an existing one, the core steps are augmented as:
  - *Design* the process to avoid the root causes of defects, and to meet customer requirements.
  - *Verify* that the process model will, in fact, avoid defects and meet customer requirements.
- This variation is sometimes called the DMADV (Define, Measure, Analyze, Design, and Verify) method.

Software Quality Assurance

Capability Maturity Model for software engineering

- Another approach to quality assurance is to objectively measure the software development processes of an organization against a "maturity model" that aims to improve the existing processes.
- The most widely used one is the *Capability Maturity Model* (*CMM*), created by the Software Development Institute in 1986 after a study of data from U.S. Department of Defense contractors.

Software Quality Assurance

Capability Maturity Model for software engineering

- CMM is based on the process maturity framework first described in the 1988 article by Watts Humphrey, "Characterizing the software process: a maturity framework" in IEEE Software. 5 (2): 73–79.
- The full representation of CMM was initiated in 1991. Version 1.1 was completed in January 1993. It was published as a book in 1995:

  Mark C. Paulk, Charles V. Weber, Bill Curtis, Mary Beth Chrissis*, The Capability Maturity Model: Guidelines for Improving the Software* Process, Addison-Wesley.

Software Quality Assurance

Capability Maturity Model for software engineering

- The term "maturity" is the degree of formality and optimization of processes, from ad hoc practices, to formally defined steps, to managed result metrics, to active optimization of the processes.
- Though the model comes from software development, it is also used to aid in business processes generally. It has also been used extensively worldwide in government, commerce, and industry.

Software Quality Assurance

Capability Maturity Model for software engineering

- A maturity model is a set of structured levels that describe how well the behaviors, practices and processes of an organization can reliably and sustainably produce required outcomes.
- A maturity model can be used as a benchmark for comparison and as an aid to understanding - for example, for comparative assessment of different organizations where there is something in common that can be used as a basis for comparison.
- In the case of the CMM, for example, the basis for comparison would be the organizations' software development processes.

Software Quality Assurance

Capability Maturity Model for software engineering

- The model involves five aspects:
  - **Maturity Levels:** a 5-level process maturity continuum - the uppermost (5th) level is the ideal state where processes would be systematically managed by a combination of process optimization and continuous process improvement.
  - **Key Process Areas:** identifies a cluster of related activities that, when performed together, achieve a set of goals considered important.
  - **Goals:** summarize the states for a key process area. It is an indicator of how much capability the organization has established at that maturity level. The goals signify the scope, boundaries, and intent of each key process area.
  - **Common Features**: include practices that implement and institutionalize a key process area: commitment to perform, ability to perform, activities performed, measurement and analysis, and verifying implementation.
  - **Key Practices:** describe infrastructure elements and practice that contribute most effectively to implementation and institutionalization of the area.

Software Quality Assurance

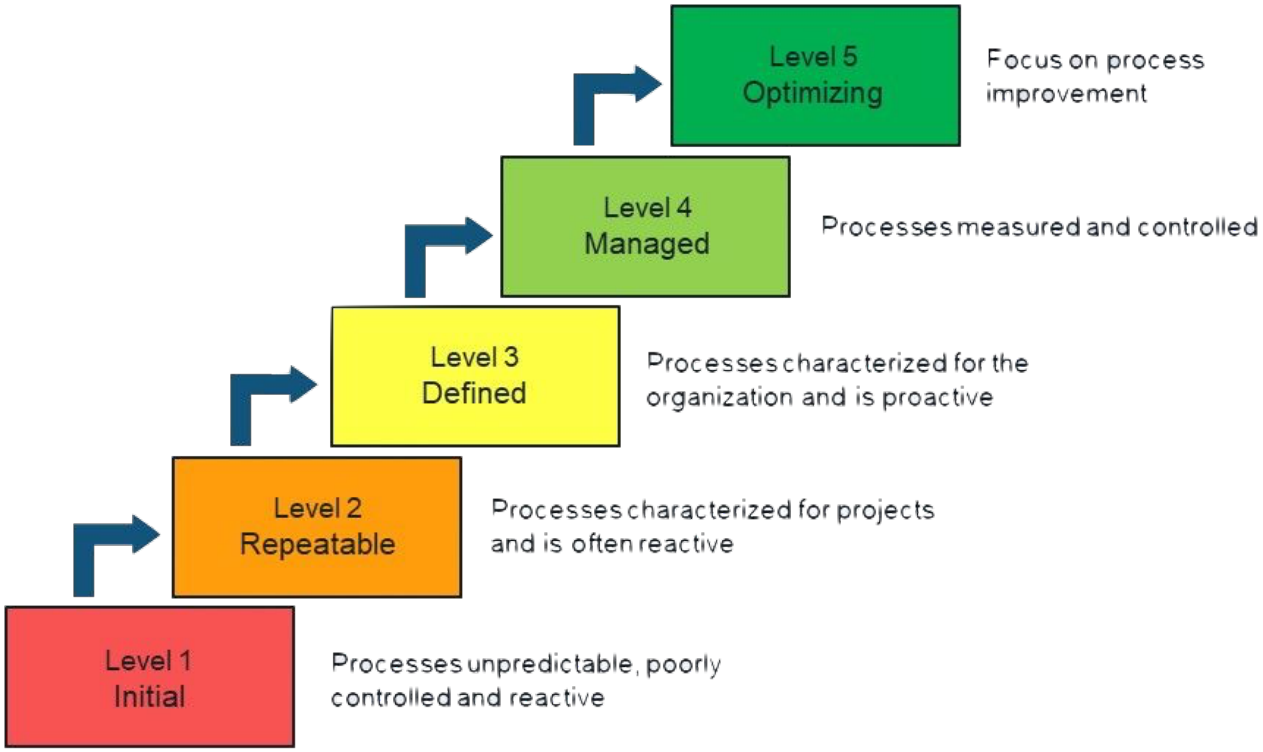
Capability Maturity Model for software engineering

- Five levels along the continuum of the model.
  1. **Initial** (chaotic, ad hoc, individual heroics) - the starting point for use of a new or undocumented repeat process.
  2. **Repeatable** - the process is at least documented sufficiently such that repeating the same steps may be attempted.
  3. **Defined** - the process is defined/confirmed as a standard business process
  4. **Managed** - the process is quantitatively managed in accordance with agreed-upon metrics.
  5. **Optimizing** - process management includes deliberate process optimization and improvement.
- Between 2008 and 2019, about 12% of appraisals given were at maturity levels 4 and 5.

Software Quality Assurance

Capability Maturity Model for software engineering

- Five levels along the continuum of the model.



Software Quality Assurance

Capability Maturity Model for software engineering

- The CMM software process framework is intended to guide those wishing to assess an organization's or project's consistency with the Key Process Areas. For each level there are five checklist types:

Type	Description
Policy	Describes the policy contents and KPA goals recommended by the Key Process Areas.
Standard	Describes the recommended content of select work products described in the Key Process Areas.
Process	Describes the process information content recommended by the Key Process Areas. These are refined into checklists for roles, entry criteria, inputs, activities, outputs, exit criteria, reviews and audits, work products managed and controlled, measurements, documented procedures, training, and tools
Procedure	Describes the recommended content of documented procedures described in the Key Process Areas.
Level overview	Provides an overview of an entire maturity level. These are further refined into checklists for key Process Areas purposes, goals, policies, and standards; process descriptions; procedures; training; tools; reviews and audits; work products; measurements

Software Quality Assurance

Capability Maturity Model for software engineering

- Application of CMM in software development has sometimes been problematic. Applying multiple models that are not integrated within and across an organization could be costly in training, appraisals, and improvement activities.
- The Capability Maturity Model Integration (CMMI) project was formed around 2002 to sort out the problem of using multiple models for software development processes.
- The CMMI model has now superseded the CMM model, though the CMM model continues to be a general theoretical process capability model used in the public domain.

Software Quality Assurance

Software reliability

- Software reliability, unlike many other quality factors, can be measured and estimated using historical and developmental data.
- Software reliability is defined in statistical terms as, “the probability of failure-free operation of a computer program in a specified environment for a specified time”
- To illustrate, program X is estimated to have a reliability of 0.999 over eight elapsed processing hours.
- In other words, if program X were to be executed 1000 times and require a total of eight hours of elapsed processing time (execution time), it is likely to operate correctly (without failure) 999 times.

Software Quality Assurance

Software reliability

- What is meant by the term *failure*? In the context of a discussion of software quality and reliability, failure is nonconformance to software requirements.
- There are gradations within this definition. Failures can be only irritating or extremely serious. One failure can be corrected within seconds, while another requires weeks or even months to correct.
- Complicating the issue even further, the correction of one failure may in fact result in the introduction of other errors that ultimately result in other failures.

Software Quality Assurance

Measures of reliability and availability

- Early work in software reliability attempted to extrapolate the mathematics of hardware reliability theory to the prediction of software reliability.
- Most hardware-related reliability models are predicated on failure due to wear rather than failure due to design defects.
- In hardware, failures due to physical wear (e.g., the effects of temperature, corrosion, shock) are more likely than a design-related failure.
- Unfortunately, the opposite is true for software. In fact, all software failures can be traced to design or implementation problems; wear does not enter the picture.

Software Quality Assurance

Measures of reliability and availability

- There has been an ongoing debate over the relationship between key concepts in hardware reliability and their applicability to software.
- Although an irrefutable link has yet to be established, it is worthwhile to consider a few simple concepts that apply to both system elements.

Software Quality Assurance

Measures of reliability and availability

- If we consider a computer-based system, a simple measure of reliability is *mean-time-between-failure (MTBF)*;
  - MTBF = MTTF + MTTR

  where the acronym *MTTF* is *mean-time-to-failure* and *MTTR* is *mean-time- to-repair*.

Software Quality Assurance

Measures of reliability and availability

- Many researchers argue that MTBF is a far more useful measure than other quality-related software metrics.
- However, MTBF can be problematic for two reasons:
  1. It projects a time span between failures, but does not provide us with a projected failure rate,
  2. It can be misinterpreted to mean average life span even though this is not what it implies.

Software Quality Assurance

Measures of reliability and availability

- An alternative measure of reliability is *failures-in-time (FIT)*—a statistical measure of how many failures a component will have over 1 billion hours of operation.
- Therefore, 1 FIT is equivalent to one failure in every billion hours of operation.

Software Quality Assurance

Measures of reliability and availability

- In addition to a reliability measure, we should also develop a measure of availability.
- Software availability is the probability that a program is operating according to requirements at a given point in time and is defined as
  - $Availability = \frac{MTTF}{(MTTF+MTTR)}$ x 100%
- The MTBF reliability measure is equally sensitive to MTTF and MTTR. The availability measure is somewhat more sensitive to MTTR, an indirect measure of the maintainability of software.

Software Quality Assurance

Software safety

- *Software safety* is a SQA activity that focuses on the identifying and assessing potential hazards that may affect software negatively and cause an entire system to fail.
- If hazards can be identified early in the software process, software design features can be specified that will either eliminate or control potential hazards.
- A modeling and analysis process is conducted as part of software safety. Initially, hazards are identified and categorized by criticality and risk.

Software Quality Assurance

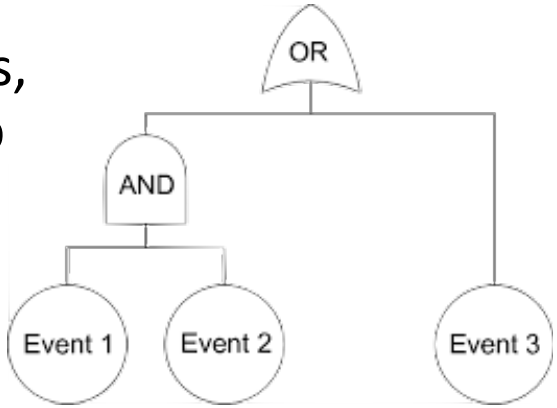
Software safety

- For example, some of the hazards associated with a computer-based cruise control for an automobile might be:
  - causes uncontrolled acceleration that cannot be stopped,
  - does not respond to depression of brake pedal (by turning off),
  - does not engage when switch is activated, and
  - slowly loses or gains speed.
- Once these system-level hazards are identified, analysis techniques are used to assign a severity and probability of occurrence.
- To be effective, software must be analyzed in the context of the entire system.

Software Quality Assurance

Software safety

- For example, a subtle user input error (people are system components) may be magnified by a software fault to produce control data that improperly positions a mechanical device.
- If and only if a set of external environmental conditions is met, the improper position of the mechanical device will cause a disastrous failure.
- Analysis techniques such as fault tree analysis, real-time logic, and Petri net models can help predict the chain of events causing a hazard and the probability that each will occur to create the chain.



Software Quality Assurance

Software safety

- Once hazards are identified and analyzed, safety-related requirements can be specified for the software.
- That is, the specification can contain a list of undesirable events and the desired system responses to these events. The role of software in managing undesirable events is then indicated.
- Software safety examines the ways in which failures result in conditions that can lead to a mishap.
- That is, failures are not considered in a vacuum, but are evaluated in the context of an entire computer-based system and its environment.

Software Quality Assurance

ISO 9000 Quality Standards

- A quality assurance system can be defined as the organizational structure, responsibilities, procedures, processes, and resources for implementing quality management.
- Quality assurance systems are created to help organizations ensure their products and services satisfy customer expectations by meeting their specifications.
- These systems cover a wide variety of activities encompassing a product's entire life cycle
- This includes planning, controlling, measuring, testing and reporting, and improving quality levels throughout the development and manufacturing process.

Software Quality Assurance

ISO 9000 Quality Standards

- *ISO 9000* from the International Standards Organization describes quality assurance elements in generic terms that can be applied to any business regardless of the products or services offered.
- To become registered to one of the quality assurance system models contained in ISO 9000, a company's quality system and operations are scrutinized by third- party auditors.
- The auditors check for compliance to the standard and for effective operation. Upon successful registration, a company is issued a certificate from a registration body represented by the auditors.
- Semiannual surveillance audits ensure continued compliance to the standard.

Software Quality Assurance

ISO 9000 Quality Standards

- Requirements delineated by ISO 9001:2015 address topics such as
  - management responsibility,
  - quality system,
  - contract review,
  - design control,
  - document and data control,
  - product identification and traceability,
  - process control,
  - inspection and testing,
  - corrective and preventive action,
  - control of quality records,
  - internal quality audits,
  - training, servicing,
  - statistical techniques.

Software Quality Assurance

ISO 9000 Quality Standards

- For a software organization to be registered to ISO 9001:2015, it must establish policies and procedures to address each of the requirements just noted (and others).
- It must then be able to demonstrate that these policies and procedures are being followed.

Software Quality Assurance

ISO 9000 Quality Standards

The ISO 9001:2015 Standard

The following outline defines the basic elements of the ISO 9001:2000 standard. Comprehensive information on the standard can be obtained from the International Organization for Standardization (**www.iso.ch**) and other Internet sources (e.g., **www.praxiom.com**).

Establish the elements of a quality management system.
- Develop, implement, and improve the system.
- Define a policy that emphasizes the importance of the system.

Document the quality system.
- Describe the process.
- Produce an operational manual.
- Develop methods for controlling (updating) documents.
- Establish methods for record keeping.

Support quality control and assurance.
- Promote the importance of quality among all stakeholders.
- Focus on customer satisfaction.
- Define a quality plan that addresses objectives, responsibilities, and authority.
- Define communication mechanisms among stakeholders.

Establish review mechanisms for the quality managemer system.
- Identify review methods and feedback mechanisms.
- Define follow-up procedures.

Identify quality resources including personnel, training, and infrastructure elements.

Establish control mechanisms.
- For planning.
- For customer requirements.
- For technical activities (e.g., analysis, design, testing).
- For project monitoring and management.

Define methods for remediation.
- Assess quality data and metrics.
- Define approach for continuous process and quality improvement.

Software Quality Assurance

SQA Plan

- An SQA Plan provides a road map for instituting software quality assurance. It is developed by the SQA group (or by the software team if an SQA group does not exist).
- An SQA plan serves as a template for SQA activities that are instituted for each software project.

Software Quality Assurance

SQA Plan Standards

- IEEE Std. 730-2014, *IEEE Standard for Software Quality Assurance* recommends a structure for a SQA test plan that identifies:
  - the purpose and scope of the plan,
  - a description of all software engineering work products (e.g. models, documents, source code) within the purview of SQA.
  - all applicable standards and practices that are applied during the software process,
  - SQA actions and tasks (including reviews and audits) and their placement throughout the software process,
  - the tools and methods that support SQA actions and tasks,
  - software configuration management procedures,
  - methods to assemble, safeguard, and maintain SQA-related records,
  - organizational roles and responsibilities relative to product quality.
- The standard is available from NEU Scholar OneSearch.