

Quality Management

- ◆ Process Quality Assurance
- ◆ Quality Reviews
- ◆ Software Standards
- ◆ Document Standards
- ◆ Software Metrics
- ◆ Product Quality Metrics

Quality Management	2
Process and Product Quality	3
Software Standards	3
Quality Reviews	4
Software Measurement and Metrics	5
Product metrics	6
Software Product Metrics	7

Quality Management

- *Software quality assurance* (often called *quality management*) is an umbrella activity that is applied throughout the software process.
- Software quality assurance (SQA) encompasses
 - ◆ an 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), and
 - ◆ measurement and reporting mechanisms
- Quality management is the application of principles, concepts, and techniques to manage and control software quality.
- Software quality management for software systems has three principal concerns:
 - ◆ At the **organizational level**, quality management is concerned with *establishing a framework of organizational processes and standards* that will lead to high quality software.
 - ◆ This means that the quality management team should take responsibility for defining the software development processes to be used and standards that should apply to the software and related documentation, including the system requirements, design, and code.
 - ◆ At the **project level**, quality management involves
 - ◆ the *application of specific quality processes*,
 - ◆ checking that these *planned processes have been followed*, and
 - ◆ ensuring that the *project outputs follow the standards* that are applicable to that project.
 - ◆ Quality management at the project level is also concerned with *establishing a quality plan for a project*.
 - ◆ The quality plan should set out the quality goals for the project and define what processes and standards are to be used.
- Quality management provides an independent check on the software development process.
- The quality management process checks the project deliverables to ensure that they are consistent with organizational standards and goals.
- The Quality Assurance team should be independent from the development team so that they can report on software quality without being influenced by software development issues.
- An independent quality management team ensures that the organizational goals of quality are not compromised by short-term budget and schedule considerations.

Process and Product Quality

- The quality of the development process directly affects the quality of delivered products. This assumption comes from manufacturing systems where product quality is intimately related to the production process.
- There is a clear link between process and product quality in manufacturing because the process is relatively easy to standardize and monitor.
- However, software is not manufactured but is designed and developed.
- Software development is a creative rather than a mechanical process, so the influence of individual skills and experience is significant.
- External factors, such as the novelty of an application or commercial pressure for an early product release, also affect product quality irrespective of the process used.
- In software development, therefore, the relationship between process quality and product quality is more complex.
- It is difficult to measure software quality attributes, such as maintainability, even after using the software for a long period.
- Consequently, it is hard to tell how process characteristics influence these attributes.
- Furthermore, because of the role of design and creativity in the software process, you can't predict how process changes will influence the quality of the product.
- However, experience has shown that process quality has a significant influence on the quality of the software.
- Process quality management and improvement can certainly lead to fewer defects in delivered software.
- Process quality management involves:
 1. Defining process standards such as how and when reviews should be conducted.
 2. Monitoring the development process to ensure that the standards are being followed.
 3. Reporting the software process to project management and to the buyer of the software

Software Standards

- Software standards play a very important role in software quality management.
- An important part of quality assurance is the definition or selection of standards that should apply to the software development process or software product.
- As part of this QA process, tools and methods to support the use of these standards may also be chosen.
- Once standards have been selected for use, project-specific processes have to be defined to monitor the use of the standards and check that they have been followed.
- There are two related types of software engineering standards that may be defined and used in software quality management:
 - ♦ **Product standards:** These apply to the software product being developed. Product standards include:
 - ♦ *document standards*, such as the structure of requirements documents,
 - ♦ *documentation standards*, such as a standard comment header for an object class definition,
 - ♦ *coding standards*, which define how a programming language should be used

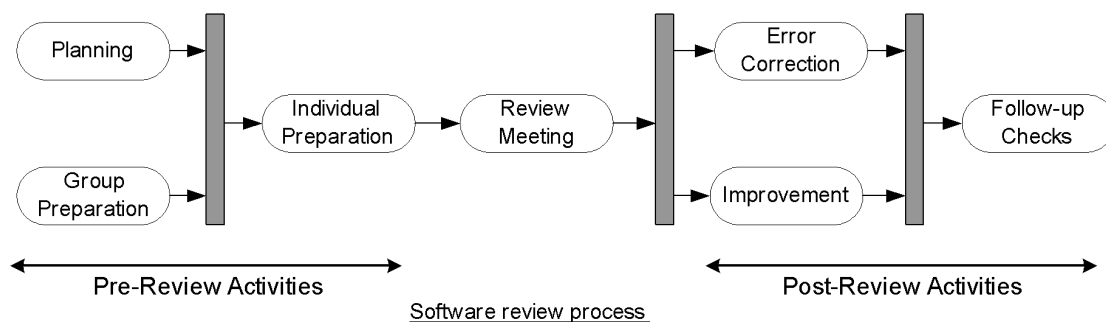
- ◆ **Process standards:** These define the processes that should be followed during software development. They should encapsulate good development practice.
 - ◆ Process standards may include definitions of specification, design and validation processes, process support tools, and a description of the documents that should be written during these processes.
- National and international bodies such as the U.S. DoD, ANSI, BSI, NATO, and the IEEE support the production of standards.
 - ◆ These are general standards that can be applied across a range of projects.
- Some organizations may require that their own standards be used in the development contracts that they place with software companies.
- National and international standards developed cover
 - ◆ software engineering terminology,
 - ◆ programming languages such as Java and C++,
 - ◆ notations such as charting symbols,
 - ◆ procedures for deriving and writing software requirements,
 - ◆ quality assurance procedures, and
 - ◆ software verification and validation processes
- More specialized standards have been developed for safety and security-critical systems.
 - ◆ such as IEC 61508
- Using international standards as a starting point, the quality assurance team should create a standards 'handbook'.
- This should define the standards that are needed by their organization.
- New standards may have to be created in response to customer or project requirements.
 - ◆ For example, standards for formal specifications may be required if these have not been used in previous projects.
- To maximize the effectiveness, quality managers who set the standards should take the following steps:
 - ◆ Involve software engineers in the selection of product standards.
 - ◆ Review and modify standards regularly to reflect changing technologies.
 - ◆ Provide software tools to support standards.

Quality Reviews

- Reviews and inspections are QA activities that check the quality of project deliverables. This involves examining the software, its documentation and records of the process to discover errors and omissions and to see if quality standards have been followed. Reviews and inspections are used alongside program testing as part of the general process of software verification and validation.
- During a review, a group of people examine the software and its associated documentation, looking for potential problems and non-conformance with standards. The review team makes informed judgments about the level of quality of a system or project deliverable.

Project managers may then use these assessments to make planning decisions and allocate resources to the development process.

- Quality reviews are based on documents that have been produced during the software development process. As well as software specifications, designs, or code, process models, test plans, configuration management procedures, process standards, and user manuals may all be reviewed. The review should check the consistency and completeness of the documents or code under review and make sure that quality standards have been followed. They are also used to help discover problems and omissions in the software or project documentation. The conclusions of the review should be formally recorded as part of the quality management process. If problems have been discovered, the reviewers' comments should be passed to the author of the software or whoever is responsible for correcting errors or omissions.
- The purpose of reviews and inspections is to improve software quality, not to assess the performance of people in the development team. Reviewing is a public process of error detection, compared with the more private component-testing process. Inevitably, mistakes that are made by individuals are revealed to the whole programming team. To ensure that all developers engage constructively with the review process, project managers have to be sensitive to individual concerns. They must develop a working culture that provides support without blame when errors are discovered.
- Normally, a review process includes following activities:
 - ♦ Planning, Group preparation Individual preparation Review meeting Error correction Improvement Follow-up checks



Software Measurement and Metrics

- Software measurement is concerned with deriving a numeric value or profile for an attribute of a software component, system, or process. By comparing these values to each other and to the standards that apply across an organization, you may be able to draw conclusions about the quality of software, or assess the effectiveness of software processes, tools, and methods.
- The long-term goal of software measurement is to use measurement in place of reviews to make judgments about software quality. Using software measurement, a system could ideally be assessed using a range of metrics and, from these measurements, a value for the quality of the system could be inferred. If the software had reached a required quality threshold, then it could be approved without review.

- When appropriate, the measurement tools might also highlight areas of the software that could be improved. However, we are still quite a long way from this ideal situation and, there are no signs that automated quality assessment will become a reality in the foreseeable future.
- Software metric is a characteristic of a software system, system documentation, or development process that can be objectively measured. Examples of metrics include the size of a product in lines of code; the Fog index (Gunning, 1962), which is a measure of the readability of a passage of written text; the number of reported faults in a delivered software product; and the number of person-days required to develop a system component.
- Software metrics may be either control metrics or predictor metrics. As the names imply, control metrics support process management, and predictor metrics help you predict characteristics of the software. Control metrics are usually associated with software processes. Examples of control or process metrics are the average effort and the time required to repair reported defects. Predictor metrics are associated with the software itself and are sometimes known as 'product metrics'. Examples of predictor metrics are the cyclomatic complexity of a module the average length of identifiers in a program, and the number of attributes and operations associated with object classes in a design.
- Managers use process measurements to decide if process changes should be made, and predictor metrics to help estimate the effort required to make software changes.
- There are two ways in which measurements of a software system may be used:
 1. To assign a value to system quality attributes by measuring the characteristics of system components, such as their cyclomatic complexity, and then aggregating these measurements, you can assess system quality attributes, such as maintainability.
 2. To identify the system components whose quality is substandard Measurements can identify individual components with characteristics that deviate from the norm. For example, you can measure components to discover those with the highest complexity. These are most likely to contain bugs because the complexity makes them harder to understand.

Product metrics

- Product metrics are predictor metrics that are used to measure internal attributes of a software system. Examples of product metrics include the system size, measured in lines of code, or the number of methods associated with each object class. Unfortunately, as I have explained earlier in this section, software characteristics that can be easily measured, such as size and cyclomatic complexity, do not have a clear and consistent relationship with quality attributes such as understandability and maintainability. The relationships vary depending on the development processes and technology used and the type of system that is being developed.
- Product metrics fall into two classes:
 1. Dynamic metrics, which are collected by measurements made of a program in execution. These metrics can be collected during system testing or after the system

has gone into use. An example might be the number of bug reports or the time taken to complete a computation.

2. Static metrics, which are collected by measurements made of representations of the system, such as the design, program, or documentation. Examples of static metrics are the code size and the average length of identifiers used.

- These types of metric are related to different quality attributes. Dynamic metrics help to assess the efficiency and reliability of a program. Static metrics help assess the complexity, understandability, and maintainability of a software system or system components.
- There is usually a clear relationship between dynamic metrics and software quality characteristics. It is fairly easy to measure the execution time required for particular functions and to assess the time required to start up a system. These relate directly to the system's efficiency. Similarly, the number of system failures and the type of failure can be logged and related directly to the reliability of the software, as discussed in Chapter 15.
- Static metrics have an indirect relationship with quality attributes. A large number of different metrics have been proposed and many experiments have tried to derive and validate the relationships between these metrics and attributes such as system complexity and maintainability. None of these experiments have been conclusive but program size and control complexity seem to be the most reliable predictors of understandability, system complexity, and maintainability.
- The metrics are applicable to any program but more specific object-oriented (OO) metrics have also been proposed. Chidamber and Kemerer's suite (CK suite) of six object oriented metrics were originally proposed in the early 1990s, they are still the most widely used OO metrics. Some UML design tools automatically collect values for these metrics as UML diagrams are created.

Software Product Metrics

Static Software Product Metrics

Software metric	Description
<i>Fan-in/Fan-out</i>	▪ Fan-in is a measure of the number of functions or methods that call another function or method (say X). ▪ Fan-out is the number of functions that are called by function X. ▪ A high value for fan-in means that X is tightly coupled to the rest of the design and changes to X will have extensive knock-on effects. ▪ A high value for fan-out suggests that the overall complexity of X may be high because of the complexity of the control logic needed to coordinate the called components.
<i>Length of code</i>	▪ This is a measure of the size of a program. ▪ Generally, the larger the size of the code of a component, the more complex and error-prone that component is likely to be. ▪ Length of code has been shown to be one of the most reliable metrics for predicting error-proneness in components.
<i>Cyclomatic complexity</i>	▪ This is a measure of the control complexity of a program.

	▪ This control complexity may be related to program understandability.
<i>Length of identifiers</i>	▪ This is a measure of the average length of identifiers (names for variables, classes, methods, etc.) in a program. ▪ The longer the identifiers, the more likely they are to be meaningful and hence the more understandable the program.
<i>Depth of conditional nesting</i>	▪ This is a measure of the depth of nesting of if-statements in a program. ▪ Deeply nested if-statements are hard to understand and potentially error-prone.
<i>Fog index</i>	▪ This is a measure of the average length of words and sentences in documents. ▪ The higher the value of a document's Fog index, the more difficult the document is to understand.

References:

- ◆ Sommerville 5th Edition, Chapter 30. [611]
 - ◆ 8th Edition, Chapter 27 (27.1)
 - ◆ 9th Edition, Chapter 24
- ◆ Software Engineering, Sommerville:
 - ◆ 9th Edition, Chapter 24 (24.1, 24.2, 24.3, 24.4)
 - ◆ 8th Edition, Chapter 27 (27.1)