Chapter 24 - Quality Management

Topics covered

- ✧ Software quality
- ✧ Software standards
- ✧ Reviews and inspections
- ✧ Quality management and agile development
- ✧ Software measurement

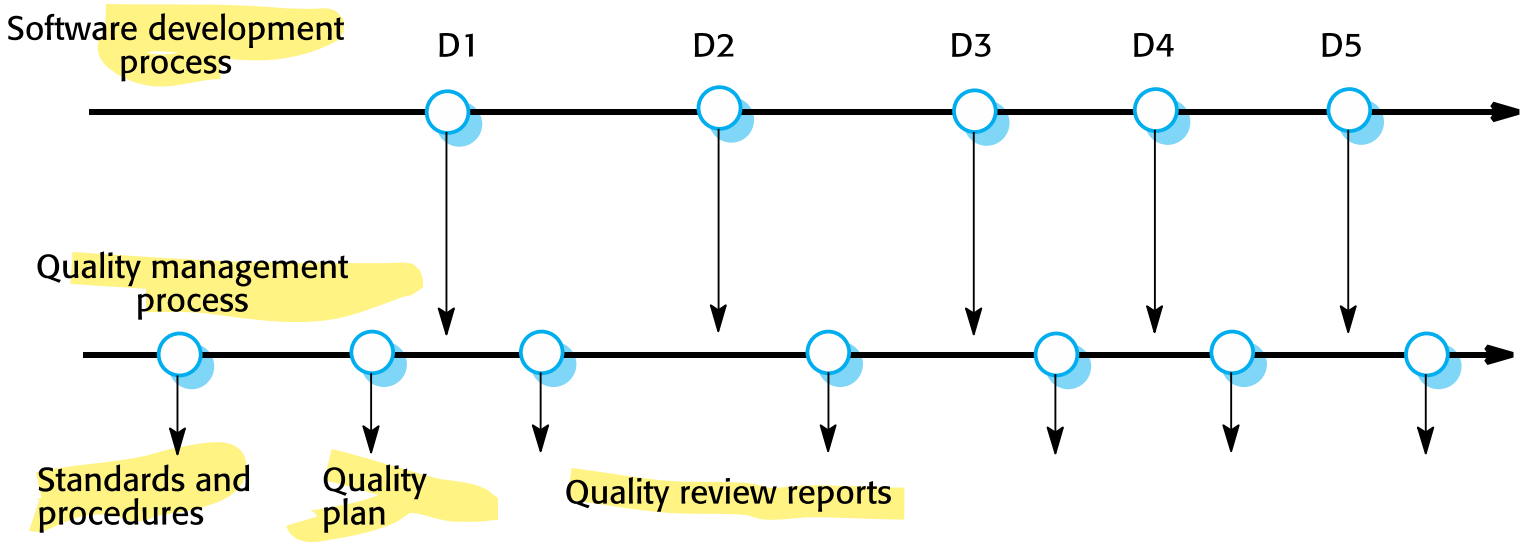
Software quality management

- Concerned with ensuring that the required level of quality is achieved in a software product.
- 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.
  - At the project level, quality management involves the application of specific quality processes and checking that these planned processes have been followed.
  - At the project level, quality management 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 activities

- 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 team should be independent from the development team so that they can take an objective view of the software. This allows them to report on software quality without being influenced by software development issues.

Quality management and software development



Quality planning

- A quality plan sets out the desired product qualities and how these are assessed and defines the most significant quality attributes.
- The quality plan should define the quality assessment process.
- It should set out which organisational standards should be applied and, where necessary, define new standards to be used.

Quality plans

- Quality plan structure
  - Product introduction;
  - Product plans;
  - Process descriptions;
  - Quality goals;
  - Risks and risk management.
- Quality plans should be short, succinct documents
  - If they are too long, no-one will read them.

Scope of quality management

- Quality management is particularly important for large, complex systems. The quality documentation is a record of progress and supports continuity of development as the development team changes.
- For smaller systems, quality management needs less documentation and should focus on establishing a quality culture.
- Techniques have to evolve when agile development is used.

Software quality

Software quality

- Quality, simplistically, means that a product should meet its specification.
- This is problematical for software systems
  - There is a tension between customer quality requirements (efficiency, reliability, etc.) and developer quality requirements (maintainability, reusability, etc.);
  - Some quality requirements are difficult to specify in an unambiguous way;
  - Software specifications are usually incomplete and often inconsistent.
- The focus may be 'fitness for purpose' rather than specification conformance.

Software fitness for purpose

- ✧ Has the software been properly tested?
- ✧ Is the software sufficiently dependable to be put into use?
- ✧ Is the performance of the software acceptable for normal use?
- ✧ Is the software usable?
- ✧ Is the software well-structured and understandable?
- ✧ Have programming and documentation standards been followed in the development process?

Non-functional characteristics

- The subjective quality of a software system is largely based on its non-functional characteristics.
- This reflects practical user experience – if the software's functionality is not what is expected, then users will often just work around this and find other ways to do what they want to do.
- However, if the software is unreliable or too slow, then it is practically impossible for them to achieve their goals.

Software quality attributes

SSRRR UTAMC PUREL

Safety	Understandability	Portability
Security	Testability	Usability
Reliability	Adaptability	Reusability
Resilience	Modularity	Efficiency
Robustness	Complexity	Learnability

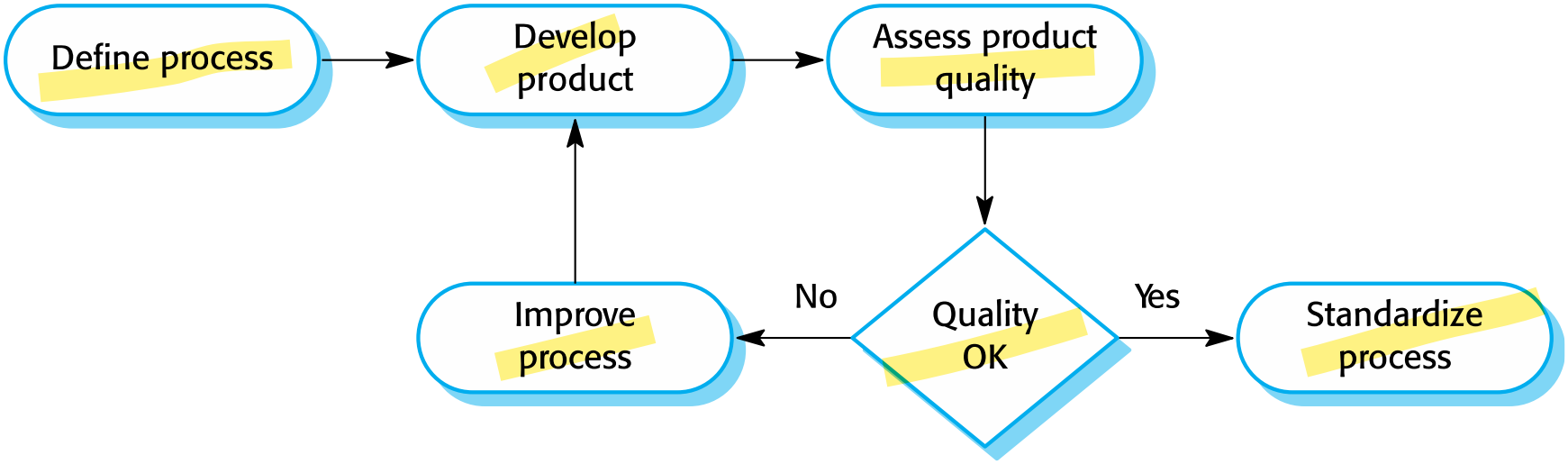
Quality conflicts

- It is not possible for any system to be optimized for all of these attributes – for example, improving robustness may lead to loss of performance.
- The quality plan should therefore define the most important quality attributes for the software that is being developed.
- The plan should also include a definition of the quality assessment process, an agreed way of assessing whether some quality, such as maintainability or robustness, is present in the product.

Process and product quality

- The quality of a developed product is influenced by the quality of the production process.
- This is important in software development as some product quality attributes are hard to assess.
- However, there is a very complex and poorly understood relationship between software processes and product quality.
  - The application of individual skills and experience is particularly important in software development;
  - External factors such as the novelty of an application or the need for an accelerated development schedule may impair product quality.

Process-based quality



Quality culture

- Quality managers should aim to develop a ‘quality culture’ where everyone responsible for software development is committed to achieving a high level of product quality.
- They should encourage teams to take responsibility for the quality of their work and to develop new approaches to quality improvement.
- They should support people who are interested in the intangible aspects of quality and encourage professional behavior in all team members.

Software standards

Software standards

- Standards define the required attributes of a product or process. They play an important role in quality management.
- Standards may be international, national, organizational or project standards.

Importance of standards

- Encapsulation of best practice- avoids repetition of past mistakes.
- They are a framework for defining what quality means in a particular setting i.e. that organization's view of quality.
- They provide continuity - new staff can understand the organisation by understanding the standards that are used.

Product and process standards

- *Product standards*
  - Apply to the software product being developed. They include document standards, such as the structure of requirements documents, documentation standards, such as a standard comment header for an object class definition, and coding standards, which define how a programming language should be used.
- *Process standards*
  - These define the processes that should be followed during software development. 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.

Product and process standards

Product standards	Process standards
Design review form	Design review conduct
Requirements document structure	Submission of new code for system building
Method header format	Version release process
Java programming style	Project plan approval process
Project plan format	Change control process
Change request form	Test recording process

Problems with standards

- They may not be seen as relevant and up-to-date by software engineers.
- They often involve too much bureaucratic form filling.
- If they are unsupported by software tools, tedious form filling work is often involved to maintain the documentation associated with the standards.

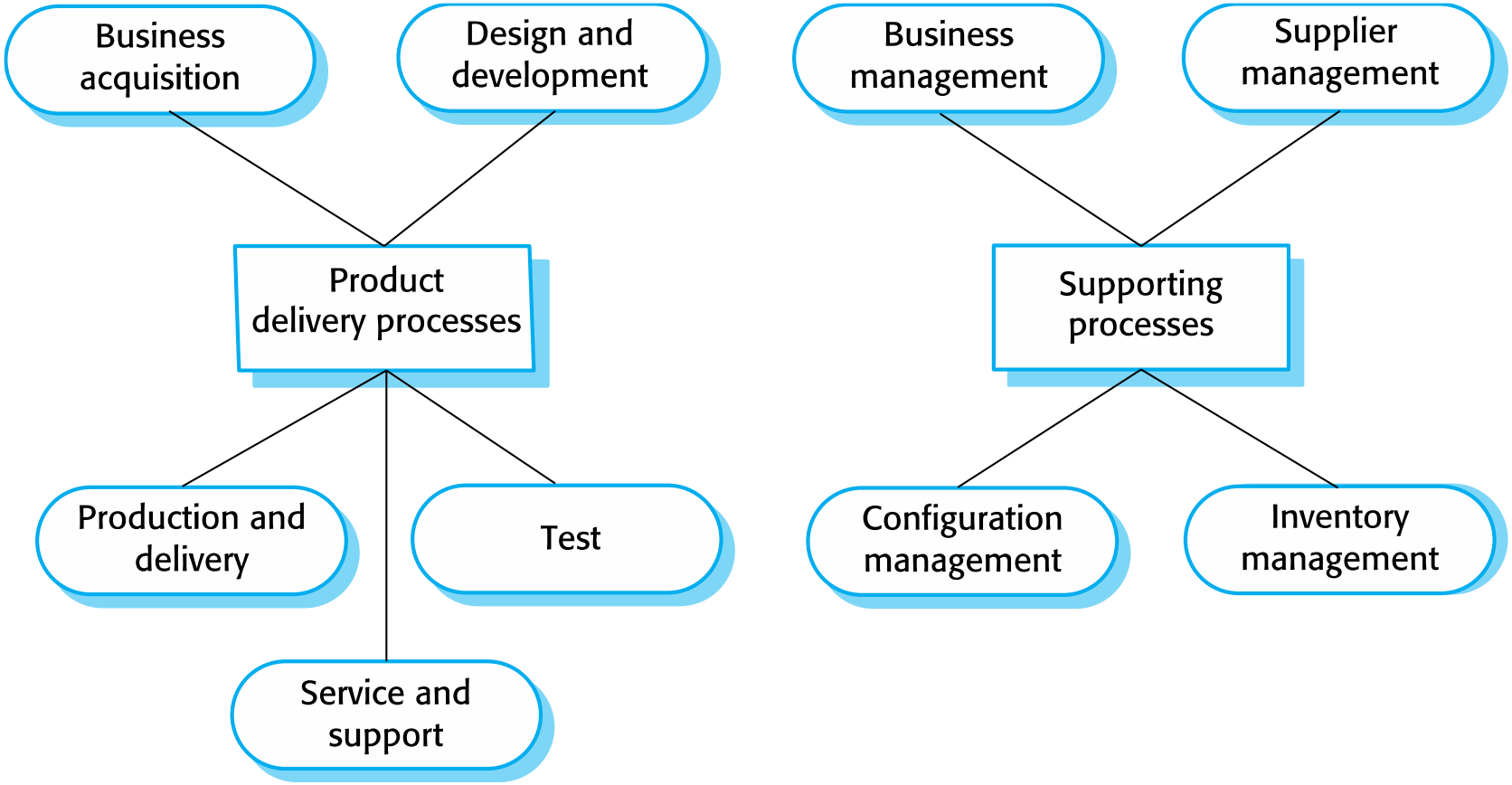
Standards development

- Involve practitioners in development. Engineers should understand the rationale underlying a standard.
- Review standards and their usage regularly. Standards can quickly become outdated and this reduces their credibility amongst practitioners.
- Detailed standards should have specialized tool support. Excessive clerical work is the most significant complaint against standards.
  - Web-based forms are not good enough.

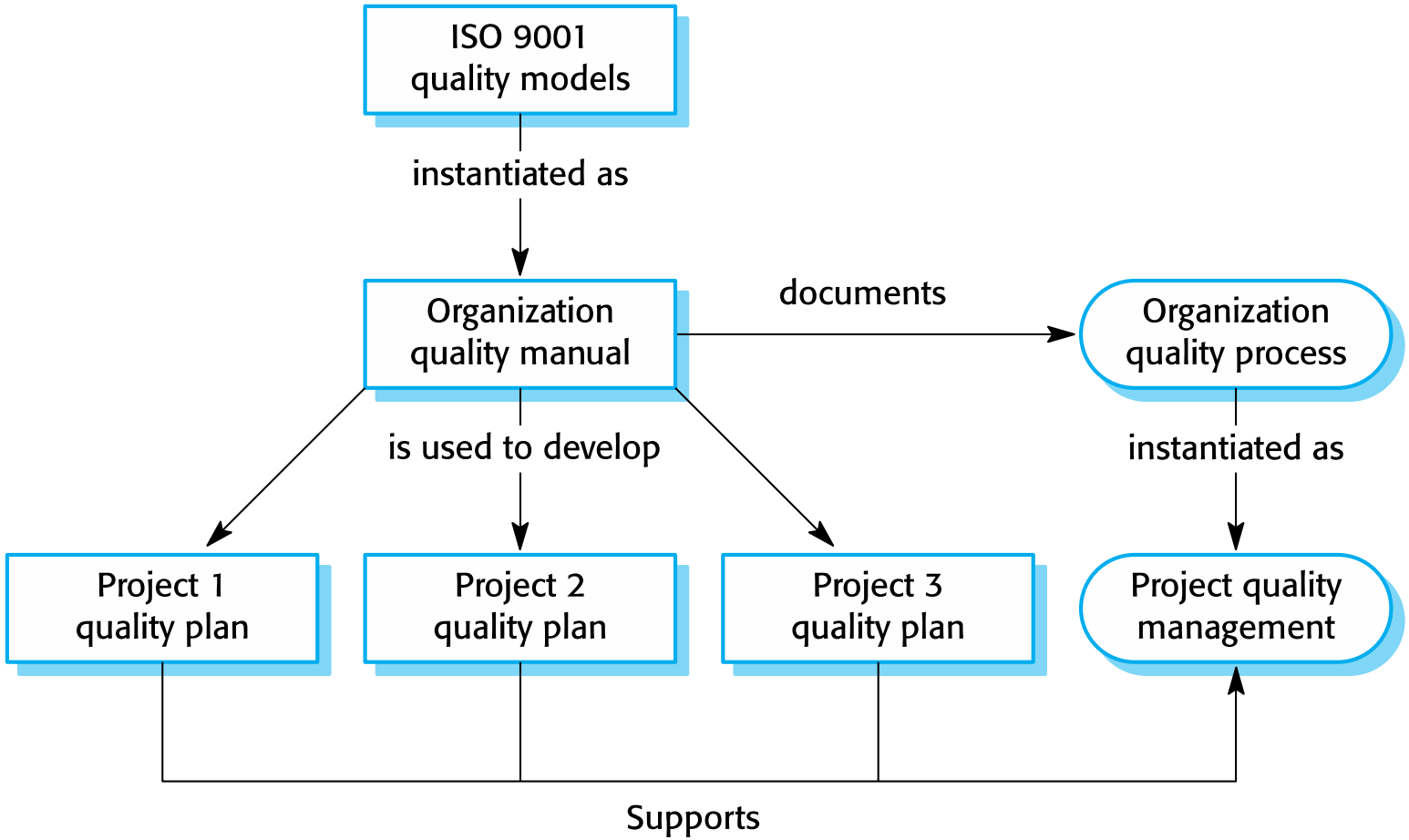
ISO 9001 standards framework

- An international set of standards that can be used as a basis for developing quality management systems.
- ISO 9001, the most general of these standards, applies to organizations that design, develop and maintain products, including software.
- The ISO 9001 standard is a framework for developing software standards.
  - It sets out general quality principles, describes quality processes in general and lays out the organizational standards and procedures that should be defined. These should be documented in an organizational quality manual.

ISO 9001 core processes



ISO 9001 and quality management



ISO 9001 certification

- Quality standards and procedures should be documented in an organisational quality manual.
- An external body may certify that an organisation's quality manual conforms to ISO 9000 standards.
- Some customers require suppliers to be ISO 9000 certified although the need for flexibility here is increasingly recognised.

Software quality and ISO9001

- The ISO 9001 certification is inadequate because it defines quality to be the conformance to standards.
- It takes no account of quality as experienced by users of the software. For example, a company could define test coverage standards specifying that all methods in objects must be called at least once.
- Unfortunately, this standard can be met by incomplete software testing that does not include tests with different method parameters. So long as the defined testing procedures are followed and test records maintained, the company could be ISO 9001 certified.

Reviews and inspections

Reviews and inspections

- A group examines part or all of a process or system and its documentation to find potential problems.
- Software or documents may be 'signed off' at a review which signifies that progress to the next development stage has been approved by management.
- There are different types of review with different objectives
  - Inspections for defect removal (product);
  - Reviews for progress assessment (product and process);
  - Quality reviews (product and standards).

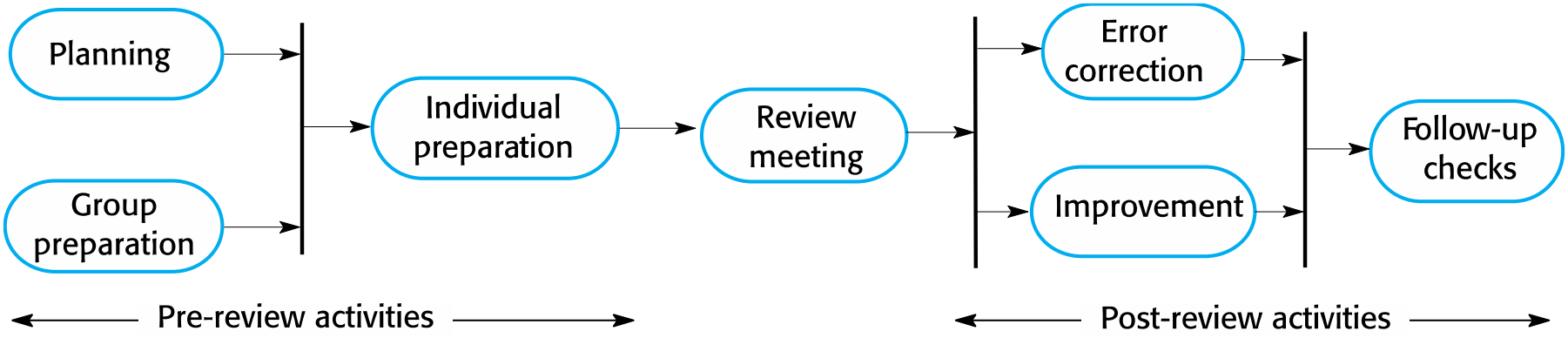
Quality reviews

- A group of people carefully examine part or all of a software system and its associated documentation.
- Code, designs, specifications, test plans, standards, etc. can all be reviewed.
- Software or documents may be 'signed off' at a review which signifies that progress to the next development stage has been approved by management.

Phases in the review process

- ✧ Pre-review activities
  - Pre-review activities are concerned with review planning and review preparation
- ✧ The review meeting
  - During the review meeting, an author of the document or program being reviewed should ‘walk through’ the document with the review team.
- ✧ Post-review activities
  - These address the problems and issues that have been raised during the review meeting.

The software review process



Distributed reviews

- The processes suggested for reviews assume that the review team has a face-to-face meeting to discuss the software or documents that they are reviewing.
- However, project teams are now often distributed, sometimes across countries or continents, so it is impractical for team members to meet face to face.
- Remote reviewing can be supported using shared documents where each review team member can annotate the document with their comments.

Program inspections

- These are peer reviews where engineers examine the source of a system with the aim of discovering anomalies and defects.
- Inspections do not require execution of a system so may be used before implementation.
- They may be applied to any representation of the system (requirements, design,configuration data, test data, etc.).
- They have been shown to be an effective technique for discovering program errors.

Inspection checklists

- Checklist of common errors should be used to drive the inspection.
- Error checklists are programming language dependent and reflect the characteristic errors that are likely to arise in the language.
- In general, the 'weaker' the type checking, the larger the checklist.
- Examples: Initialisation, Constant naming, loop termination, array bounds, etc.

An inspection checklist (a)

Fault class	Inspection check
Data faults	• Are all program variables initialized before their values are used? • Have all constants been named? • Should the upper bound of arrays be equal to the size of the array or Size -1? • If character strings are used, is a delimiter explicitly assigned? • Is there any possibility of buffer overflow?
Control faults	• For each conditional statement, is the condition correct? • Is each loop certain to terminate? • Are compound statements correctly bracketed? • In case statements, are all possible cases accounted for? • If a break is required after each case in case statements, has it been included?
Input/output faults	• Are all input variables used? • Are all output variables assigned a value before they are output? • Can unexpected inputs cause corruption?

An inspection checklist (b)

Fault class	Inspection check
Interface faults	• Do all function and method calls have the correct number of parameters? • Do formal and actual parameter types match? • Are the parameters in the right order? • If components access shared memory, do they have the same model of the shared memory structure?
Storage management faults	• If a linked structure is modified, have all links been correctly reassigned? • If dynamic storage is used, has space been allocated correctly? • Is space explicitly deallocated after it is no longer required?
Exception management faults	• Have all possible error conditions been taken into account?

Quality management and agile development

Quality management and agile development

- Quality management in agile development is informal rather than document-based.
- It relies on establishing a quality culture, where all team members feel responsible for software quality and take actions to ensure that quality is maintained.
- The agile community is fundamentally opposed to what it sees as the bureaucratic overheads of standards-based approaches and quality processes as embodied in ISO 9001.

Shared good practice

- *Check before check-in*
  - Programmers are responsible for organizing their own code reviews with other team members before the code is checked in to the build system.
- *Never break the build*
  - Team members should not check in code that causes the system to fail. Developers have to test their code changes against the whole system and be confident that these work as expected.
- *Fix problems when you see them*
  - If a programmer discovers problems or obscurities in code developed by someone else, they can fix these directly rather than referring them back to the original developer.

Reviews and agile methods

- The review process in agile software development is usually informal.
- In Scrum,, there is a review meeting after each iteration of the software has been completed (a sprint review), where quality issues and problems may be discussed.
- In Extreme Programming, pair programming ensures that code is constantly being examined and reviewed by another team member.

Pair programming

- This is an approach where 2 people are responsible for code development and work together to achieve this.
- Code developed by an individual is therefore constantly being examined and reviewed by another team member.
- Pair programming leads to a deep knowledge of a program, as both programmers have to understand the program in detail to continue development.
- This depth of knowledge is difficult to achieve in inspection processes and pair programming can find bugs that would not be discovered in formal inspections.

Pair programming weaknesses

- *Mutual misunderstandings*
  - Both members of a pair may make the same mistake in understanding the system requirements. Discussions may reinforce these errors.
- *Pair reputation*
  - Pairs may be reluctant to look for errors because they do not want to slow down the progress of the project.
- *Working relationships*
  - The pair's ability to discover defects is likely to be compromised by their close working relationship that often leads to reluctance to criticize work partners.

Agile QM and large systems

- When a large system is being developed for an external customer, agile approaches to quality management with minimal documentation may be impractical.
  - If the customer is a large company, it may have its own quality management processes and may expect the software development company to report on progress in a way that is compatible with them.
  - Where there are several geographically distributed teams involved in development, perhaps from different companies, then informal communications may be impractical.
  - For long-lifetime systems, the team involved in development will changeWithout documentation, new team members may find it impossible to understand development.

Software measurement

Software measurement

- Software measurement is concerned with deriving a numeric value for an attribute of a software product or process.
- This allows for objective comparisons between techniques and processes.
- Although some companies have introduced measurement programmes, most organisations still don't make systematic use of software measurement.
- There are few established standards in this area.

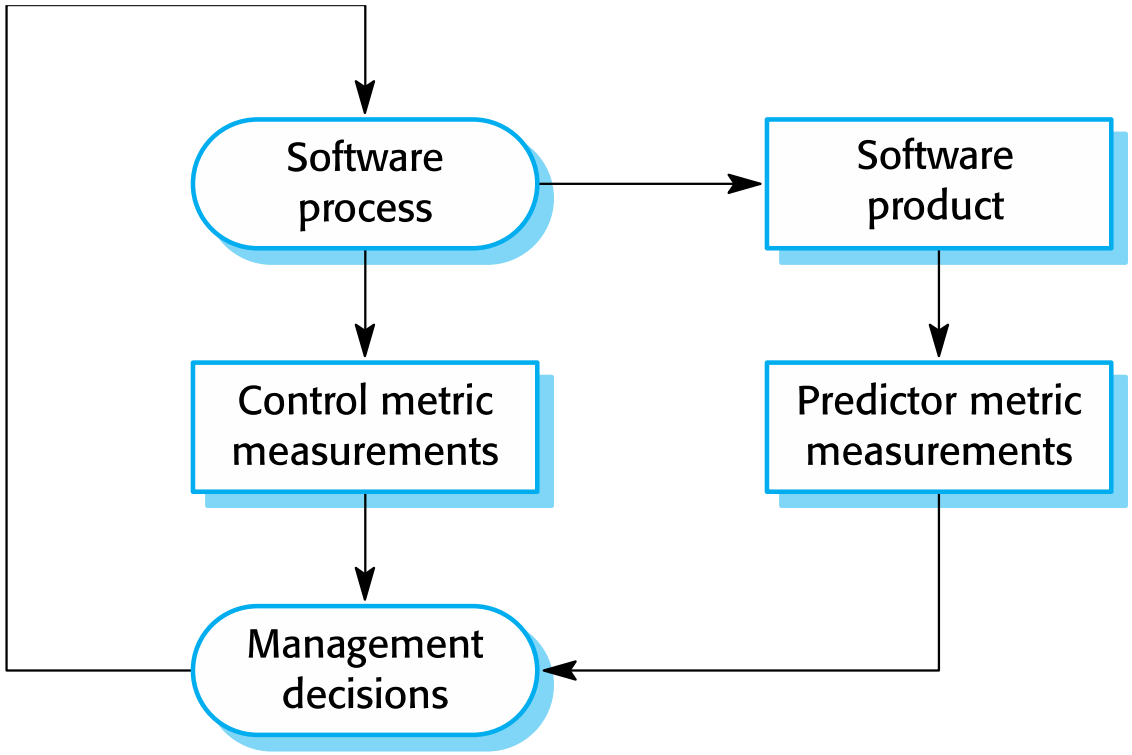
Software metric

- Any type of measurement which relates to a software system, process or related documentation
  - Lines of code in a program, the Fog index, number of person-days required to develop a component.
- Allow the software and the software process to be quantified.
- May be used to predict product attributes or to control the software process.
- Product metrics can be used for general predictions or to identify anomalous components.

Types of process metric

- *The time taken for a particular process to be completed*
  - This can be the total time devoted to the process, calendar time, the time spent on the process by particular engineers, and so on.
- *The resources required for a particular process*
  - Resources might include total effort in person-days, travel costs or computer resources.
- *The number of occurrences of a particular event*
  - Examples of events that might be monitored include the number of defects discovered during code inspection, the number of requirements changes requested, the number of bug reports in a delivered system and the average number of lines of code modified in response to a requirements change.

Predictor and control measurements



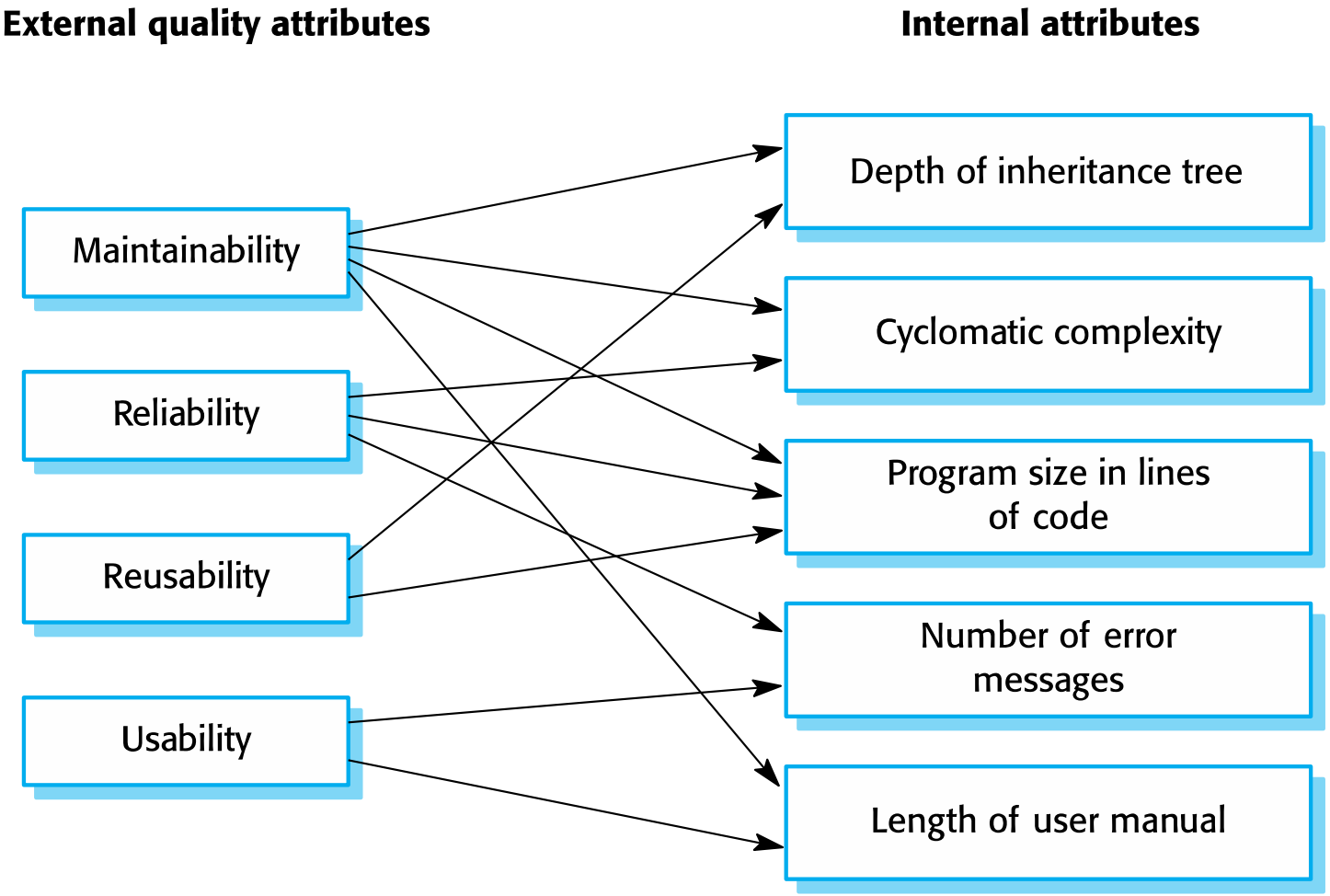
Use of measurements

- 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.
- To identify the system components whose quality is sub-standard
  - 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.

Metrics assumptions

- A software property can be measured accurately.
- The relationship exists between what we can measure and what we want to know. We can only measure internal attributes but are often more interested in external software attributes.
- This relationship has been formalised and validated.
- It may be difficult to relate what can be measured to desirable external quality attributes.

Relationships between internal and external software



Problems with measurement in industry

- It is impossible to quantify the return on investment of introducing an organizational metrics program.
- There are no standards for software metrics or standardized processes for measurement and analysis.
- In many companies, software processes are not standardized and are poorly defined and controlled.
- Most work on software measurement has focused on code-based metrics and plan-driven development processes. However, more and more software is now developed by configuring ERP systems or COTS.
- Introducing measurement adds additional overhead to processes.

Empirical software engineering

- Software measurement and metrics are the basis of empirical software engineering.
- This is a research area in which experiments on software systems and the collection of data about real projects has been used to form and validate hypotheses about software engineering methods and techniques.
- Research on empirical software engineering, this has not had a significant impact on software engineering practice.
- It is difficult to relate generic research to a project that is different from the research study.

Product metrics

- A quality metric should be a predictor of product quality.
- Classes of product metric
  - Dynamic metrics which are collected by measurements made of a program in execution;
  - Static metrics which are collected by measurements made of the system representations;
  - Dynamic metrics help assess efficiency and reliability
  - Static metrics help assess complexity, understandability and maintainability.

Dynamic and static metrics

- Dynamic metrics are closely related to software quality attributes
  - It is relatively easy to measure the response time of a system (performance attribute) or the number of failures (reliability attribute).
- Static metrics have an indirect relationship with quality attributes
  - You need to try and derive a relationship between these metrics and properties such as complexity, understandability and maintainability.

Static software product metrics

Software metric	Description
Fan-in/Fan-out	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.
Length of code	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.

Static software product metrics

Software metric	Description
Cyclomatic complexity	This is a measure of the control complexity of a program. This control complexity may be related to program understandability. I discuss cyclomatic complexity in Chapter 8.
Length of identifiers	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.
Depth of conditional nesting	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.
Fog index	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.

The CK object-oriented metrics suite

Object-oriented metric	Description
Weighted methods per class (WMC)	This is the number of methods in each class, weighted by the complexity of each method. Therefore, a simple method may have a complexity of 1, and a large and complex method a much higher value. The larger the value for this metric, the more complex the object class. Complex objects are more likely to be difficult to understand. They may not be logically cohesive, so cannot be reused effectively as superclasses in an inheritance tree.
Depth of inheritance tree (DIT)	This represents the number of discrete levels in the inheritance tree where subclasses inherit attributes and operations (methods) from superclasses. The deeper the inheritance tree, the more complex the design. Many object classes may have to be understood to understand the object classes at the leaves of the tree.
Number of children (NOC)	This is a measure of the number of immediate subclasses in a class. It measures the breadth of a class hierarchy, whereas DIT measures its depth. A high value for NOC may indicate greater reuse. It may mean that more effort should be made in validating base classes because of the number of subclasses that depend on them.

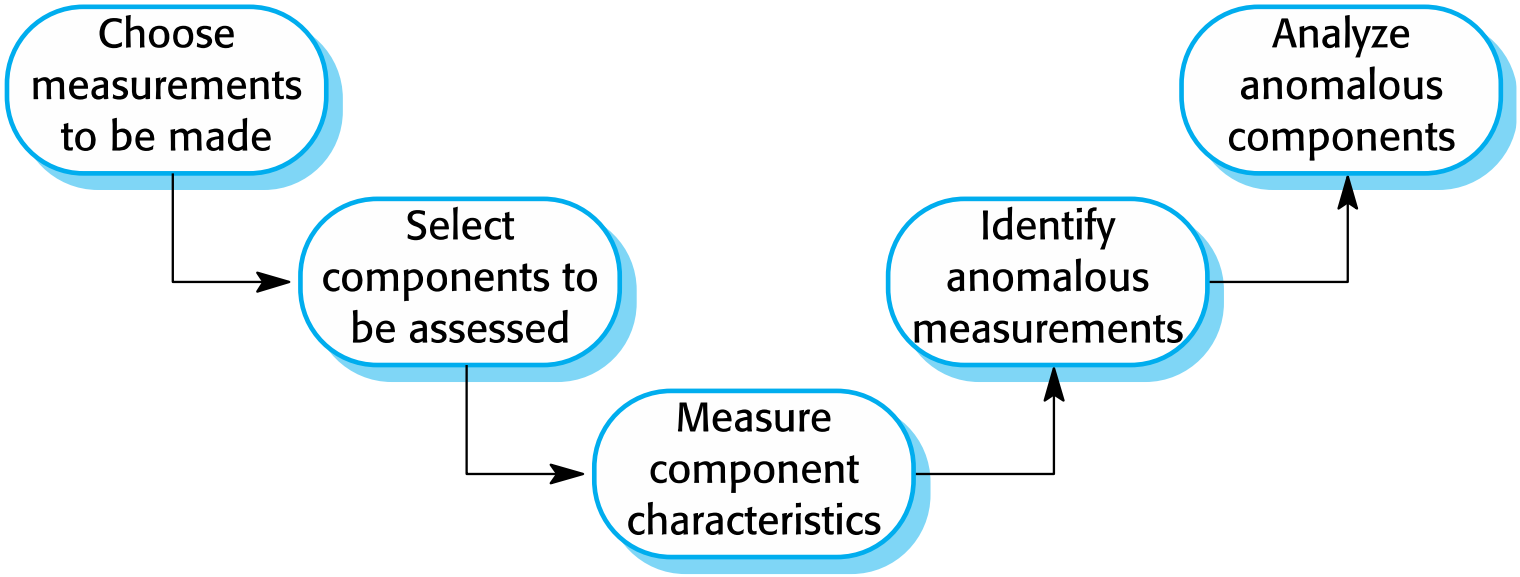
The CK object-oriented metrics suite

Object-oriented metric	Description
Coupling between object classes (CBO)	Classes are coupled when methods in one class use methods or instance variables defined in a different class. CBO is a measure of how much coupling exists. A high value for CBO means that classes are highly dependent, and therefore it is more likely that changing one class will affect other classes in the program.
Response for a class (RFC)	RFC is a measure of the number of methods that could potentially be executed in response to a message received by an object of that class. Again, RFC is related to complexity. The higher the value for RFC, the more complex a class and hence the more likely it is that it will include errors.
Lack of cohesion in methods (LCOM)	LCOM is calculated by considering pairs of methods in a class. LCOM is the difference between the number of method pairs without shared attributes and the number of method pairs with shared attributes. The value of this metric has been widely debated and it exists in several variations. It is not clear if it really adds any additional, useful information over and above that provided by other metrics.

Software component analysis

- System component can be analyzed separately using a range of metrics.
- The values of these metrics may then compared for different components and, perhaps, with historical measurement data collected on previous projects.
- Anomalous measurements, which deviate significantly from the norm, may imply that there are problems with the quality of these components.

The process of product measurement



Measurement ambiguity

- When you collect quantitative data about software and software processes, you have to analyze that data to understand its meaning.
- It is easy to misinterpret data and to make inferences that are incorrect.
- You cannot simply look at the data on its own. You must also consider the context where the data is collected.

Measurement surprises

- Reducing the number of faults in a program leads to an increased number of help desk calls
  - The program is now thought of as more reliable and so has a wider more diverse market. The percentage of users who call the help desk may have decreased but the total may increase;
  - A more reliable system is used in a different way from a system where users work around the faults. This leads to more help desk calls.

Software context

- Processes and products that are being measured are not insulated from their environment.
- The business environment is constantly changing and it is impossible to avoid changes to work practice just because they may make comparisons of data invalid.
- Data about human activities cannot always be taken at face value. The reasons why a measured value changes are often ambiguous. These reasons must be investigated in detail before drawing conclusions from any measurements that have been made.

Software analytics

- *Software analytics is analytics on software data for managers and software engineers with the aim of empowering software development individuals and teams to gain and share insight from their data to make better decisions.*

Software analytics enablers

- The automated collection of user data by software product companies when their product is used.
  - If the software fails, information about the failure and the state of the system can be sent over the Internet from the user's computer to servers run by the product developer.
- The use of open source software available on platforms such as Sourceforge and GitHub and open source repositories of software engineering data.
  - The source code of open source software is available for automated analysis and this can sometimes be linked with data in the open source repository.

Analytics tool use

- Tools should be easy to use as managers are unlikely to have experience with analysis.
- •Tools should run quickly and produce concise outputs rather than large volumes of information.
- •Tools should make many measurements using as many parameters as possible. It is impossible to predict in advance what insights might emerge.
- •Tools should be interactive and allow managers and developers to explore the analyses.

Status of software analytics

- Software analytics is still immature and it is too early to say what effect it will have.
- Not only are there general problems of 'big data' processing, our knowledge depends on collected data from large companies.
  - This is primarily from software products and it is unclear if the tools and techniques that are appropriate for products can also be used with custom software.
- Small companies are unlikely to invest in the data collection systems that are required for automated analysis so may not be able to use software analytics.

Key points

- Software quality management is concerned with ensuring that software has a low number of defects and that it reaches the required standards of maintainability, reliability, portability etc. Software standards are important for quality assurance as they represent an identification of ‘best practice’. When developing software, standards provide a solid foundation for building good quality software.
- Reviews of the software process deliverables involve a team of people who check that quality standards are being followed. Reviews are the most widely used technique for assessing quality.

Key points

- In a program inspection or peer review, a small team systematically checks the code. They read the code in detail and look for possible errors and omissions. The problems detected are discussed at a code review meeting.
- Agile quality management relies on establishing a quality culture where the development team works together to improve software quality.
- Software measurement can be used to gather quantitative data about software and the software process.

Key points

- You may be able to use the values of the software metrics that are collected to make inferences about product and process quality.
- Product quality metrics are particularly useful for highlighting anomalous components that may have quality problems. These components should then be analyzed in more detail.
- Software analytics is the automated analysis of large volumes of software product and process data to discover relationships that may provide insights for project managers and developers.