Course Lecture Schedule

Maria
Susan
Sami
Hyrynsalmi

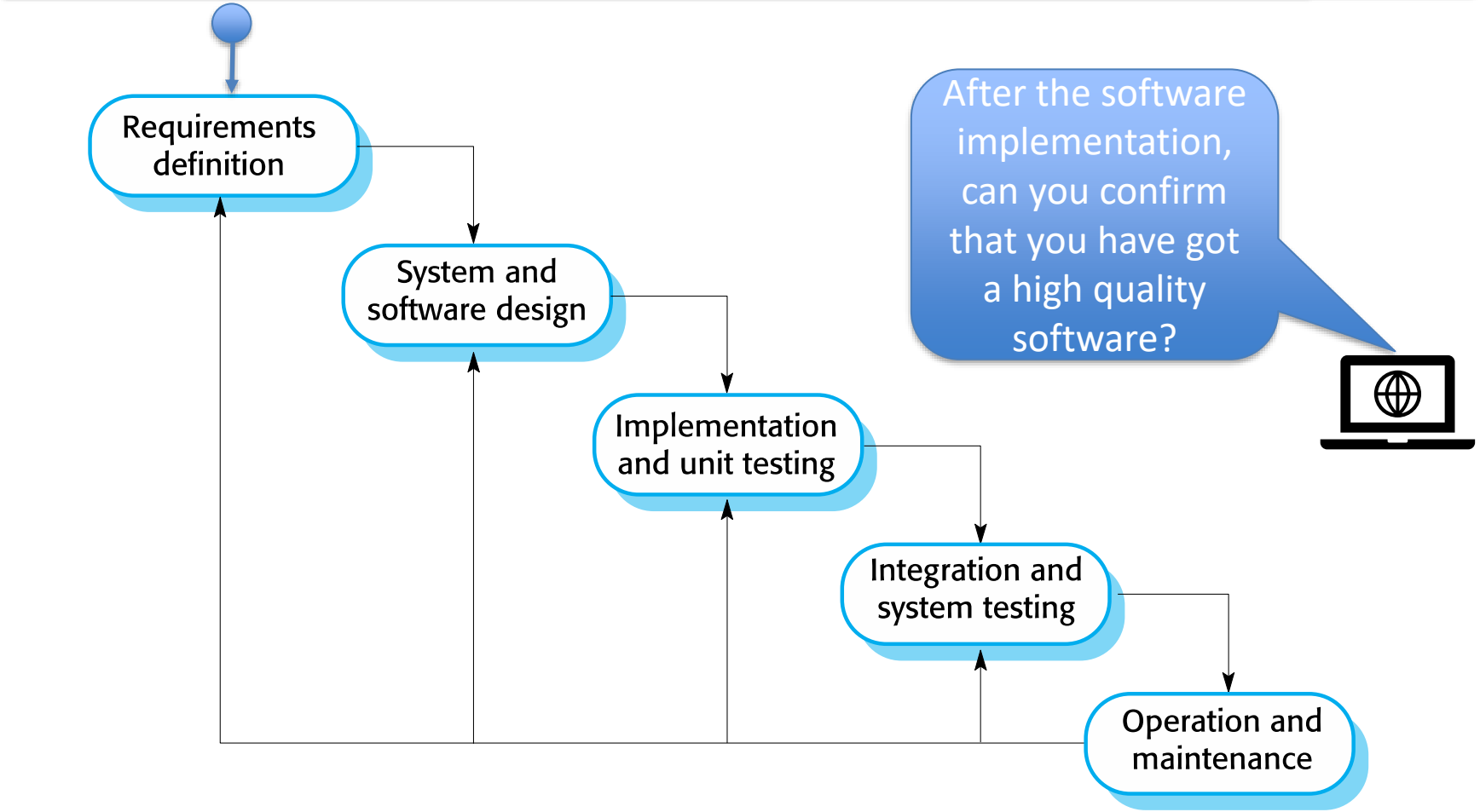
Date	Topic	Book Chapter(s)
Wed 8.9.	Course introduction	
Tue 14.9.	Introduction to Software Engineering	Chapter 1
Tue 21.9.	Software Processes	Chapter 2
Mon 27.9	Agile Software Engineering	Chapter 3
Tue 5.10.	Requirements Engineering	Chapter 4
Mon 11.10.	Architectural Design	Chapter 6
Wed 20.10.	Modeling and implementation	Chapters 5 & 7
Mon 1.11.	**Testing & Quality**	**Chapters 8 & 24**
Mon 8.11.	Software Evolution & Configuration Management	Chapters 9 & 25
Mon 15.11.	Software Project Management	Chapter 22
Mon 22.11.	Software Project Planning	Chapter 23
Mon 29.11.	Global Software Engineering	
Wed 8.12.	Software Business	
Mon 13.12.	Last topics	

Lecture 8
Testing & Quality

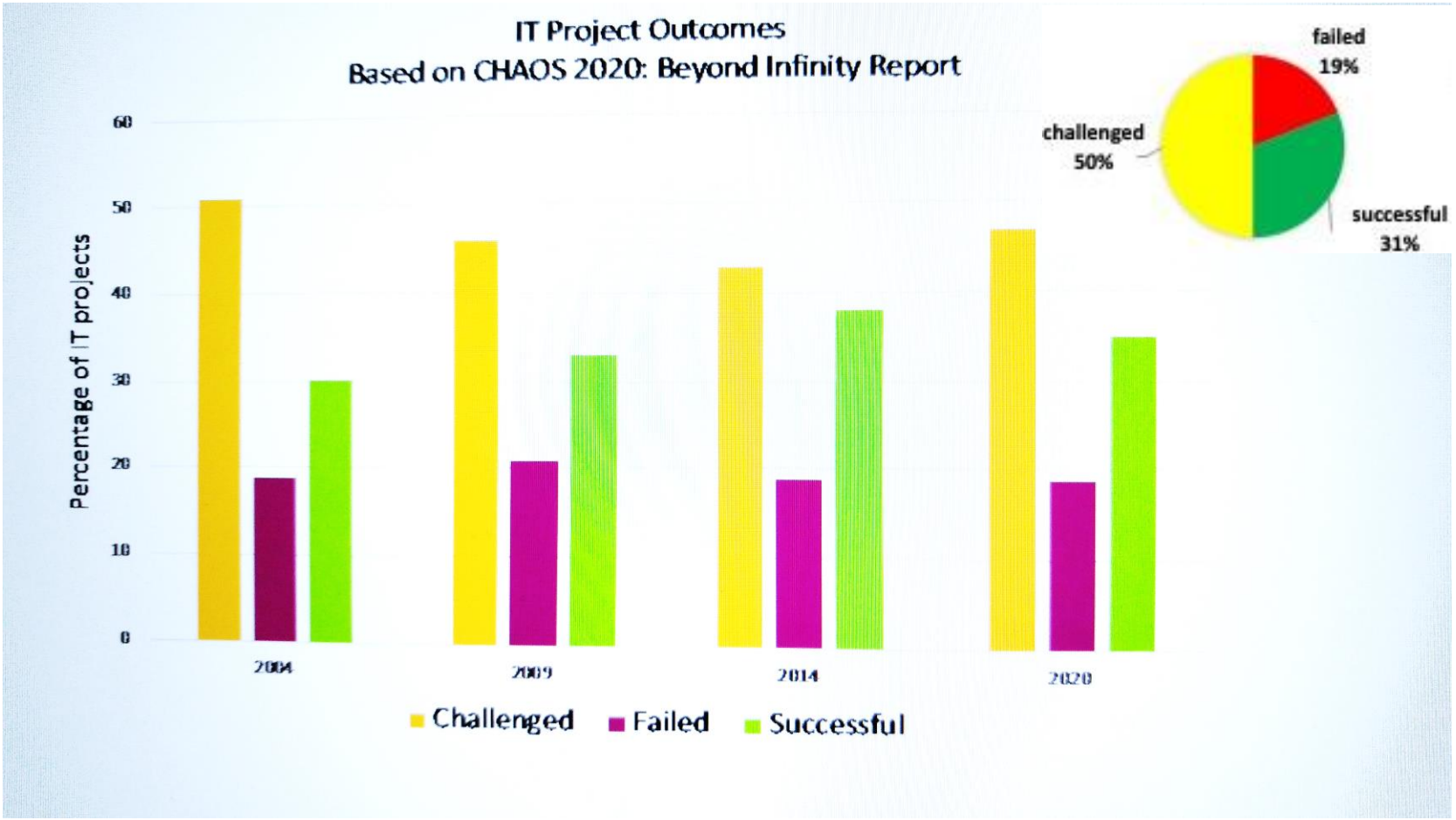
Topics covered

- ✧ Software quality
- ✧ Reviews and inspections
- ✧ Testing
  - Unit testing
  - Component testing
  - System testing
  - Test-driven development
  - Release testing
  - User testing
  - Test-driven development

Testing stage in waterfall model



Chaos Report 2020



The Cost of Poor Software Quality (CPSQ) in the US

- The largest contributor to CPSQ is operational software failures. For 2020 estimated a 22% growth over 2 years . The underlying cause is primarily unmitigated flaws in the software.
- The next largest contributor to CPSQ in unsuccessful development projects rose by 46% since 2018. The underlying causes are varied, but one consistent theme has been the lack of attention to quality.
- Legacy system problems contributed down from $635 B in 2018, mostly still due to non-value added "waste."

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.

Software quality attributes

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

- The subjective quality of a software system is largely based on its non-functional characteristics.
- This reflects practical user experience -if the software is unreliable or too slow, then it is practically impossible for them to achieve their goals.
- There may be conflicts between quality attributes– for example, improving robustness may lead to loss of performance.

Scope of quality management

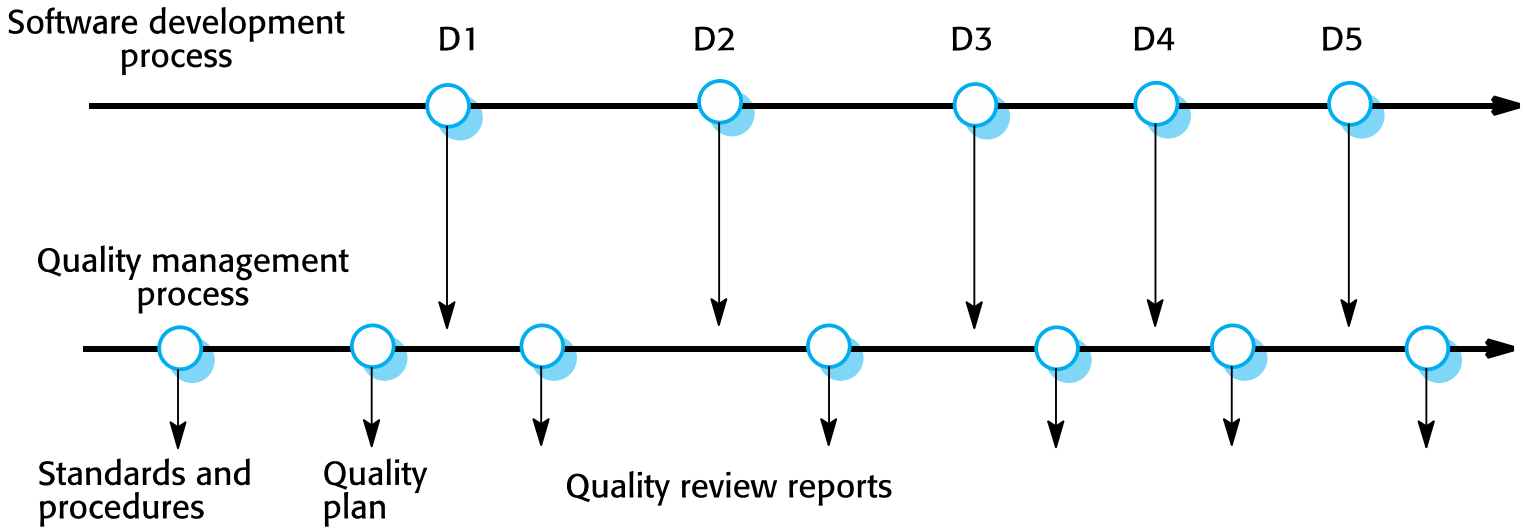
- Quality management is concerned with ensuring that the required level of quality is achieved in a software product.
  - Quality management is particularly important for large, complex systems. The quality documentation is a record of progress and supports continuity of development as the development team changes.
  - For smaller systems, quality management needs less documentation and should focus on establishing a quality culture.

Software quality management

- Quality management provides an independent check by an independent quality team on the software development process and the project deliverables to ensure that they are consistent with organizational standards and goals.
- Three principal concerns:
  - Organizational level-establishing a framework of organizational processes.
  - Project level-the application of specific quality processes.
  - Project level-establishing a quality plan for a project about quality goals, processes and standards.

Quality management and software development

The QM team checks the project deliverables to ensure they are consistent with organizational stands and goals.



Quality planning

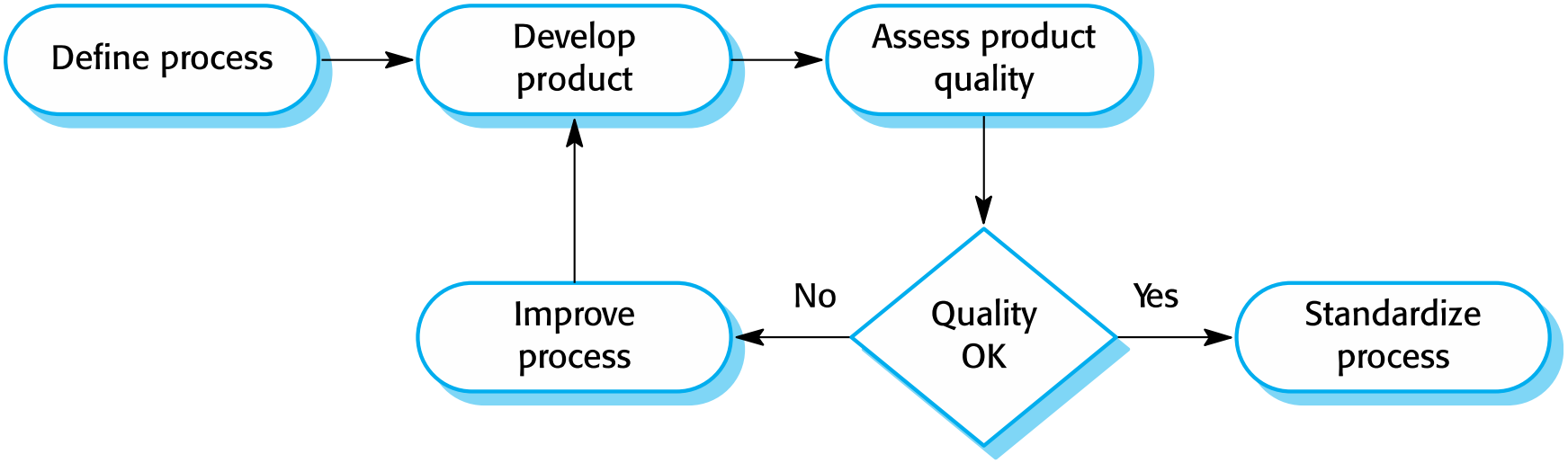
- A quality plan sets out the desired product qualities and how these are assessed and defines the most significant quality attributes.
- Quality plan structure
  - Product introduction;
  - Product plans;
  - Process descriptions;
  - Quality goals;
  - Risks and risk management.
- Quality plans should be short, succinct documents

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

✧ The quality of product is influenced by the quality of the production process.

Process-based quality



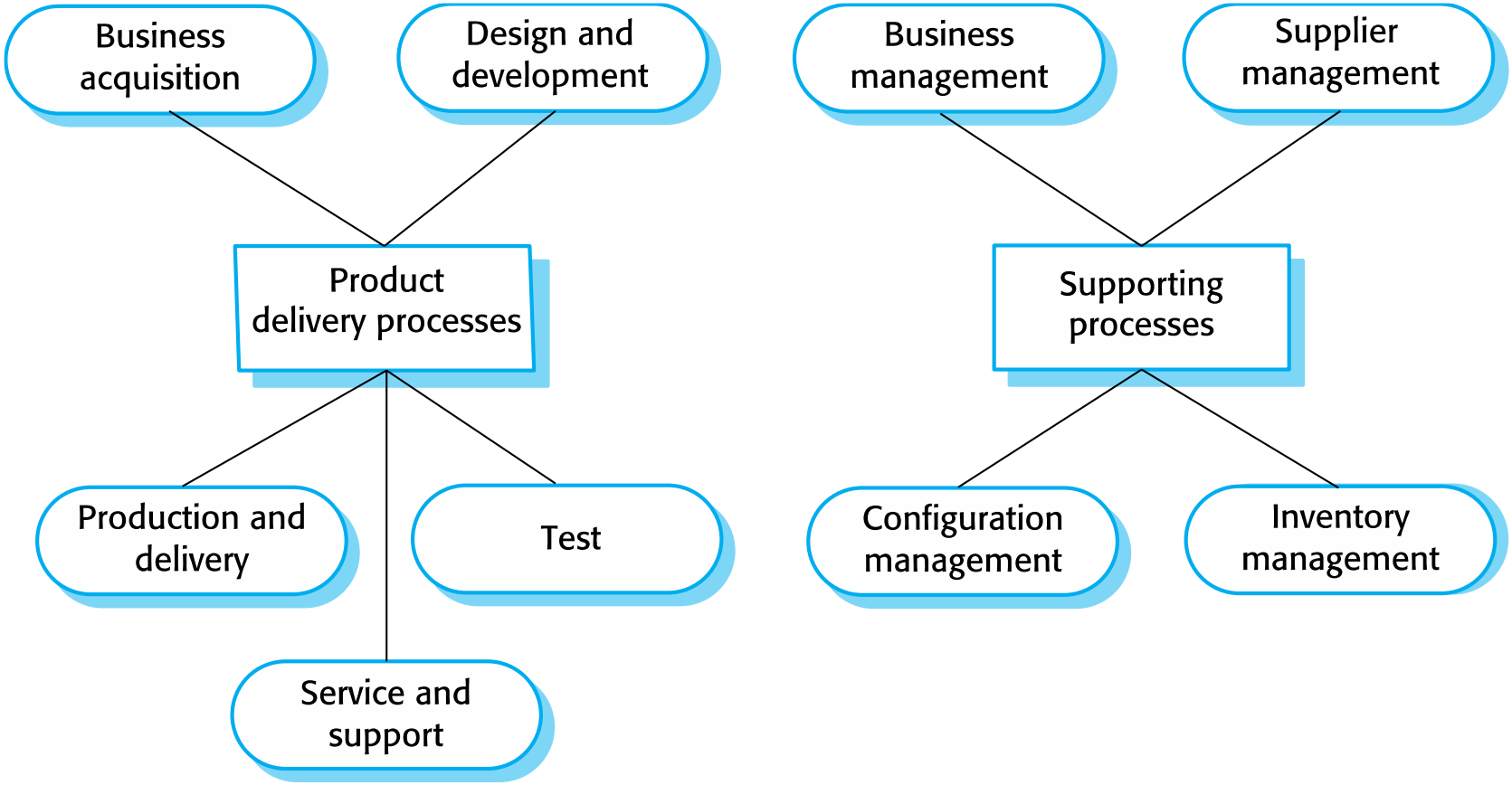
Software standards

- Standards define the required attributes of a product or process.
- 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.
- Standards may be international, national, organizational or project standards.

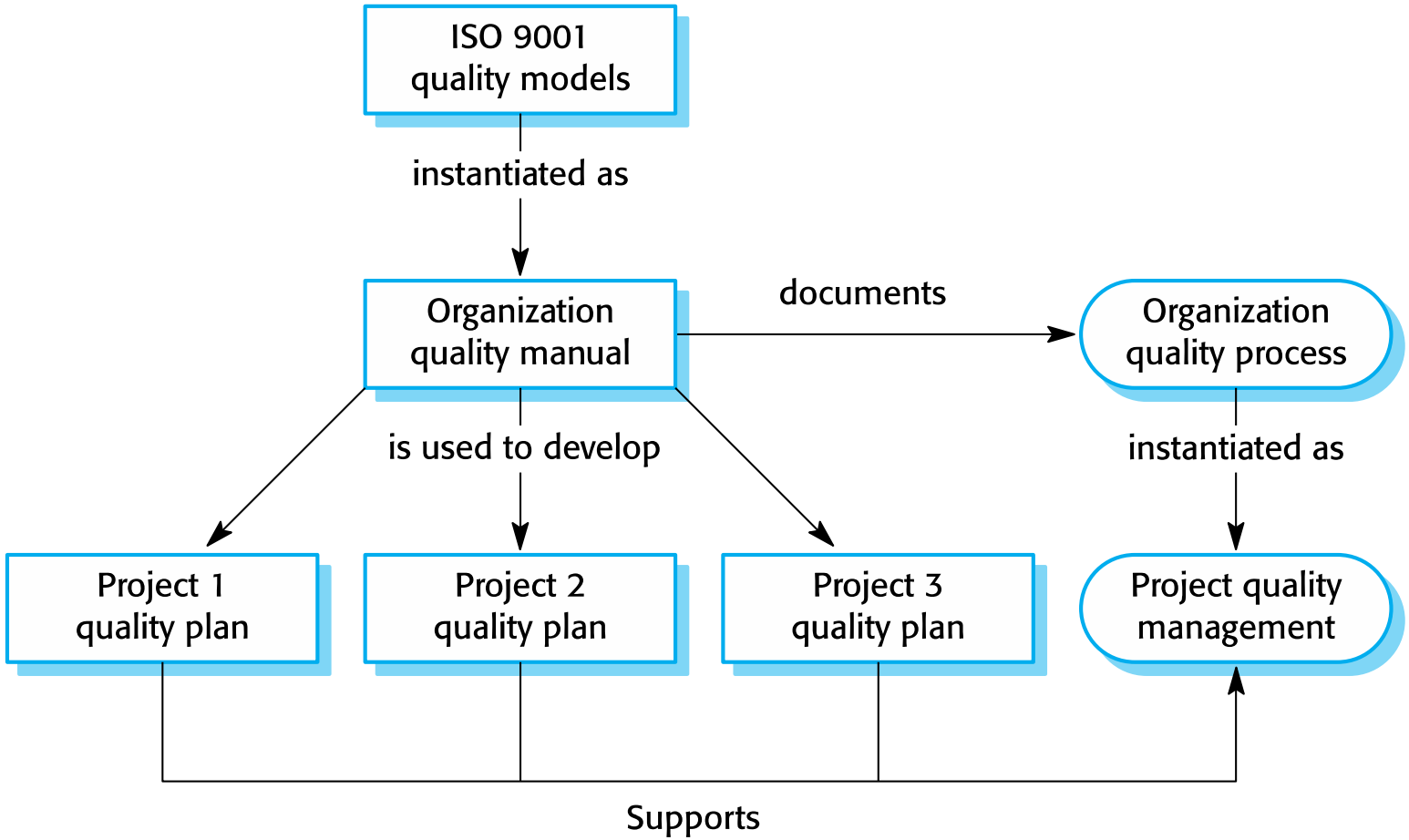
ISO 9001 standards framework

- Some international set of standards 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.
  - general quality principles
  - quality processes in general
  - organizational standards and procedures that should be defined

ISO 9001 core processes



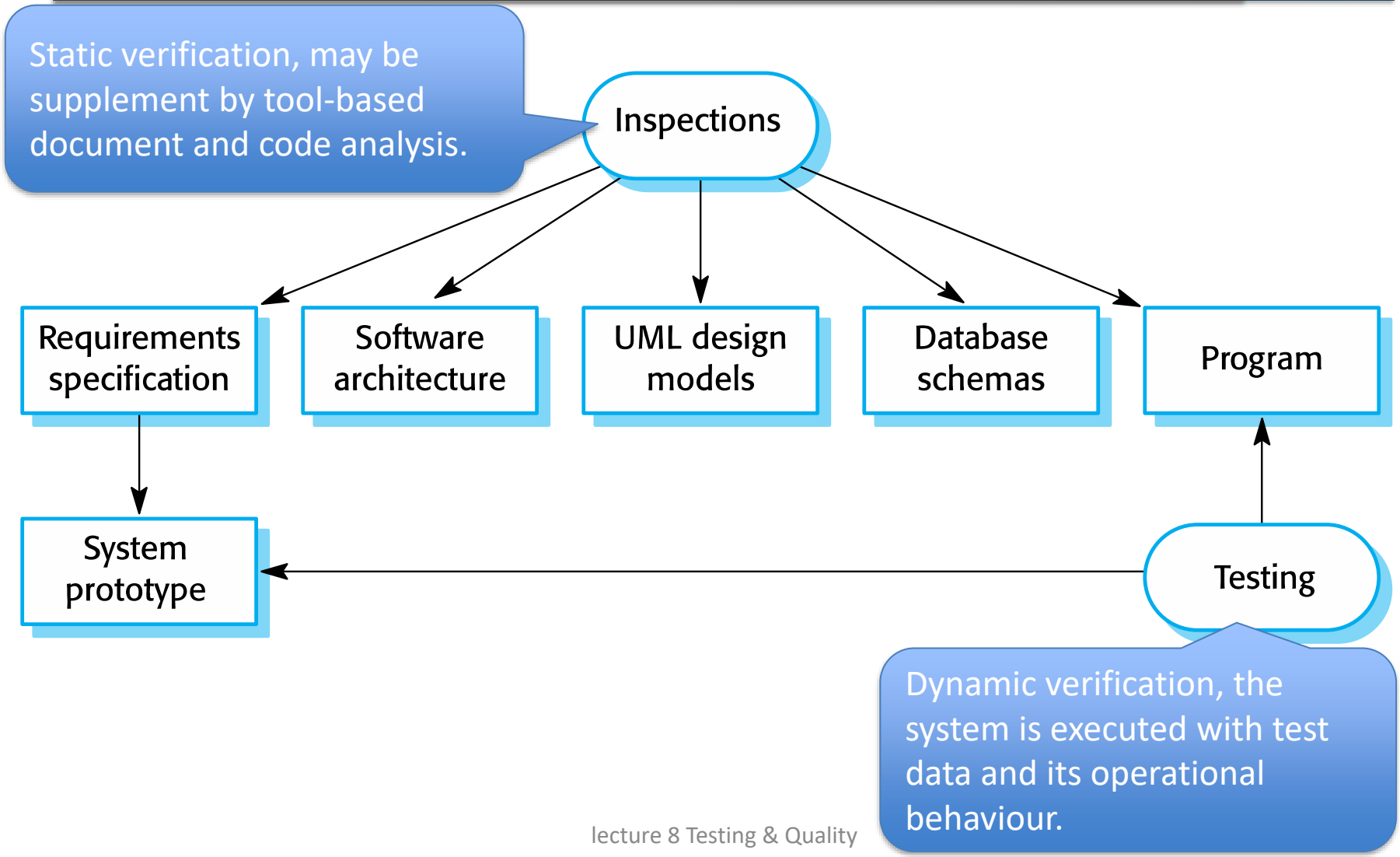
ISO 9001 and quality management



Verification vs validation

- Aim of V & V is to establish confidence that the system is ‘fit for purpose’(Depends on system’s purpose, user expectations and marketing environment).
- Verification: "Are we building the product right?”.
  - The software should conform to its specification.
- Validation: "Are we building the right product?”.
  - The software should do what the user really requires.

Inspections and testing concerns



Inspections and testing during V&V

- Inspections and testing are complementary and not opposing verification techniques.
- Both should be used during the V & V process.
- Inspections can check conformance with a specification but not conformance with the customer's real requirements.
- Inspections cannot check non-functional characteristics such as performance, usability, etc.

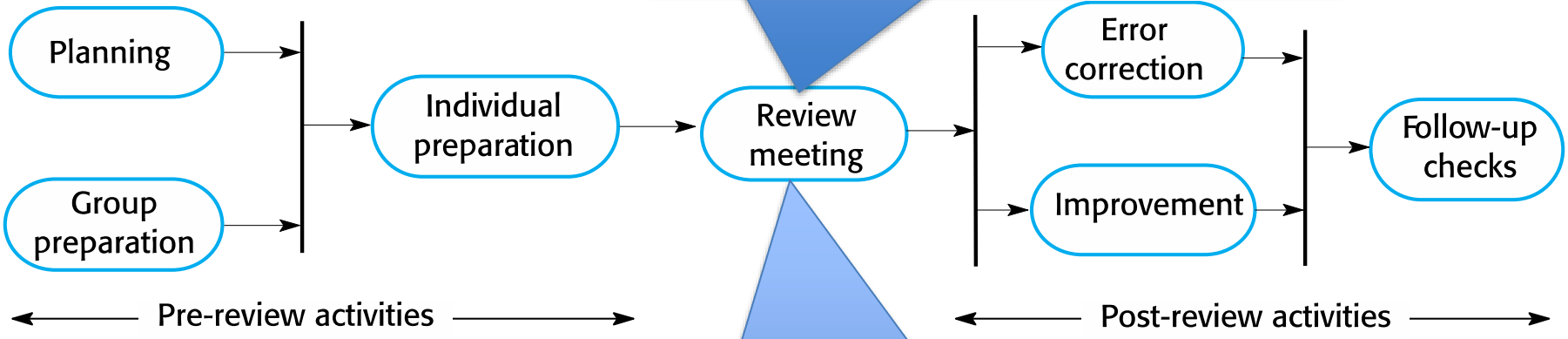
Reviews and inspections

Quality reviews and inspections

- A group of people carefully examine part or all of a software system and its associated documentation.
- Not require execution of a system so may be used before implementation.
- May be applied to any representation of the system (requirements, designs, configuration data, test data, etc.).
- Have been shown to be an effective technique for discovering program errors.
- Software or documents may be 'signed off' at a review which signifies that progress to the next development stage has been approved by management.

The software review process

Project teams are now often distributed, remote reviewing can be supported using shared documents where each review team member can annotate the document with their comments.



During the review meeting, an author of the document or program being reviewed should 'walk through' the document with the review team.

Advantages of inspections

- During testing, errors can mask (hide) other errors. Because inspection is a static process, you don't have to be concerned with interactions between errors.
- Incomplete versions of a system can be inspected without additional costs.
- As well as searching for program defects, an inspection can also consider broader quality attributes of a program, such as compliance with standards, portability and maintainability.

An inspection checklist (a)

✧ Checklist of common errors should be used to drive the inspection.

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?

✧ Error checklists are programming language dependent. In general, the 'weaker' the type checking, the larger the checklist.


Software Engineering
Ian Sommerville

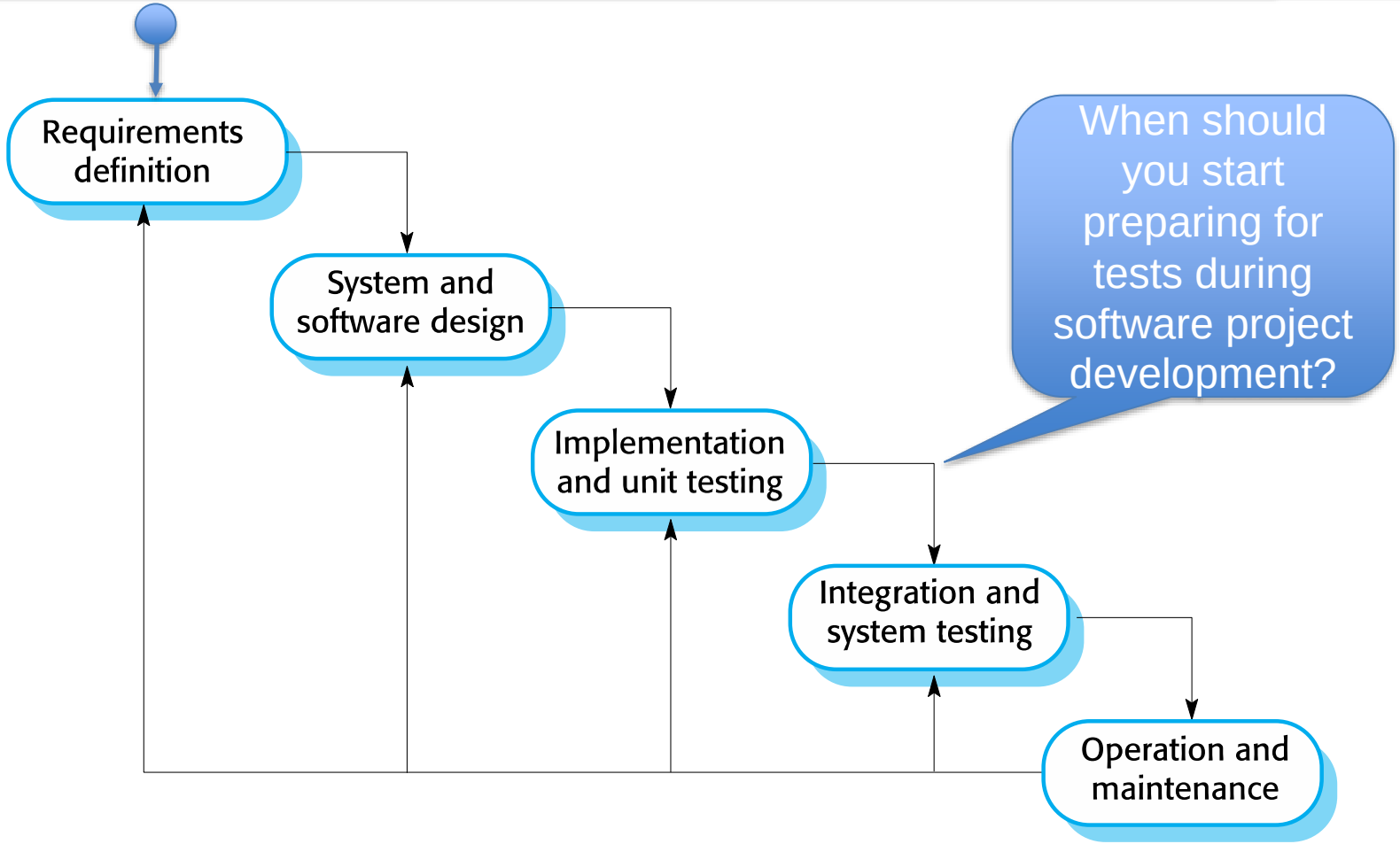

1903
HEBEI UNIVERSITY OF TECHNOLOGY

Testing

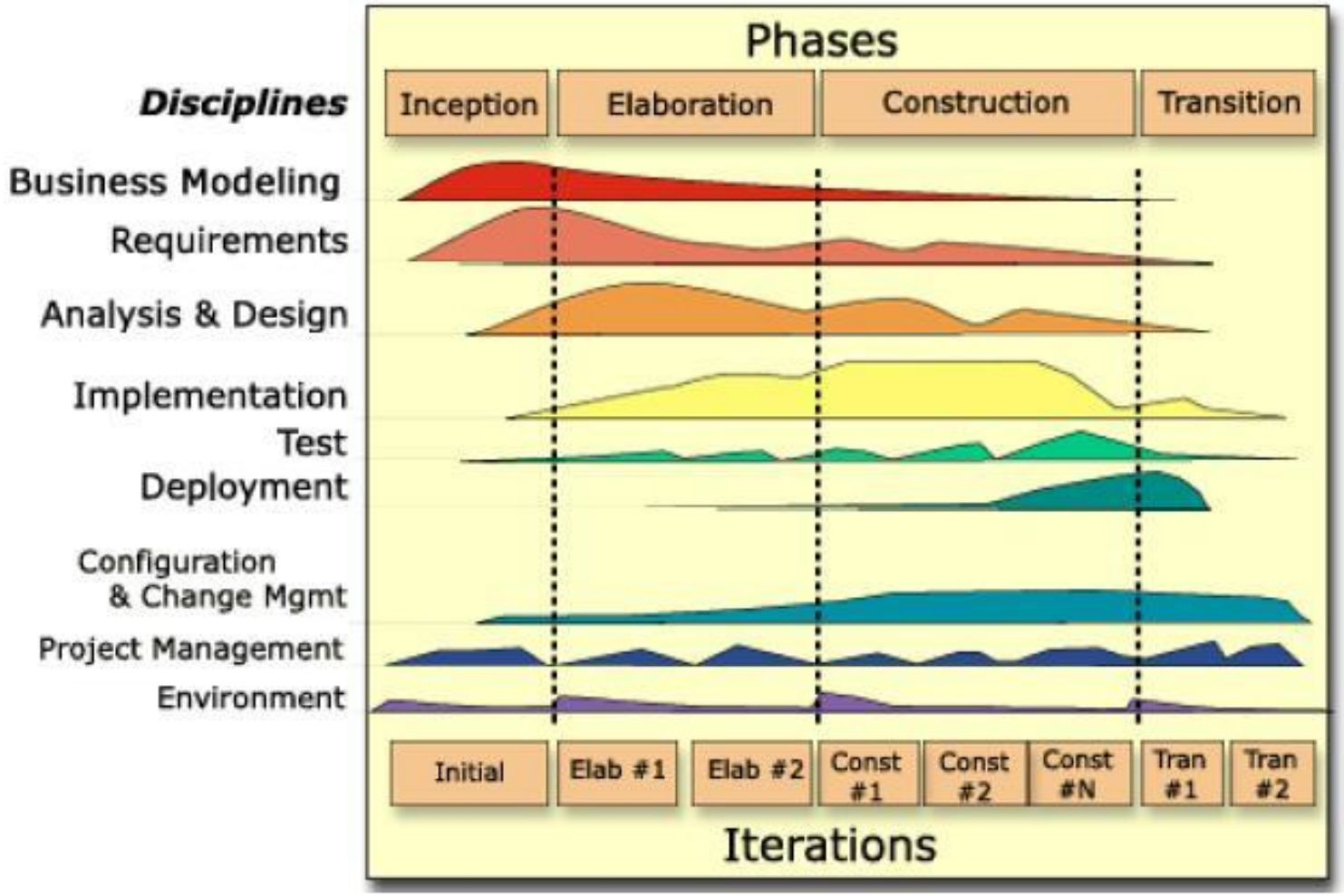
Program testing

- When you test software, you execute a program using artificial data.
- You check the results of the test run for errors, anomalies or information about the program's non-functional attributes.
- Can reveal the presence of errors NOT their absence.

Testing stage in waterfall model



Testing & Quality in RUP model



Testing process goals

✧ Validation testing

- To demonstrate to the developer and the system customer that the software meets its requirements.
- A successful test shows that the system operates as intended.

✧ Defect testing

- To discover faults or defects in the software where its behavior is incorrect or not in conformance with its specification.
- A successful test is a test that makes the system perform incorrectly and so exposes a defect in the system.

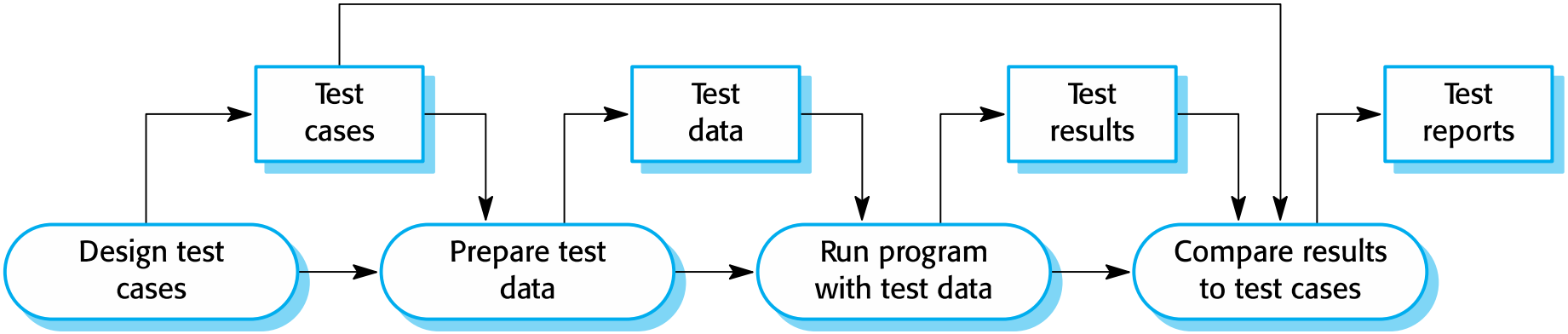
Stages of testing

- Development testing（T1-T3）, where the system is tested during development to discover bugs and defects.
- Release testing（T4）, where a separate testing team test a complete version of the system before it is released to users.
- User testing（T5）, where users or potential users of a system test the system in their own environment.

Development testing

- Development testing includes all testing activities that are carried out by the team developing the system.
  - Unit testing（T1）, where individual program units or object classes are tested, should focus on testing the functionality of objects or methods.
  - Component testing（T2）, where several individual units are integrated to create composite components, should focus on testing component interfaces.
  - System testing（T3）, where some or all of the components in a system are integrated and the system is tested as a whole, should focus on testing component interactions.

A model of the software testing process



T1-Unit testing

- Unit testing is the process of testing individual components in isolation.
- Units may be:
  - Individual functions or methods within an object
  - Object classes with several attributes and methods
  - Composite components with defined interfaces used to access their functionality.
- Unit testing is usually a white-box testing process where tests are derived from internal structure, design and coding of software.

Choosing unit test cases

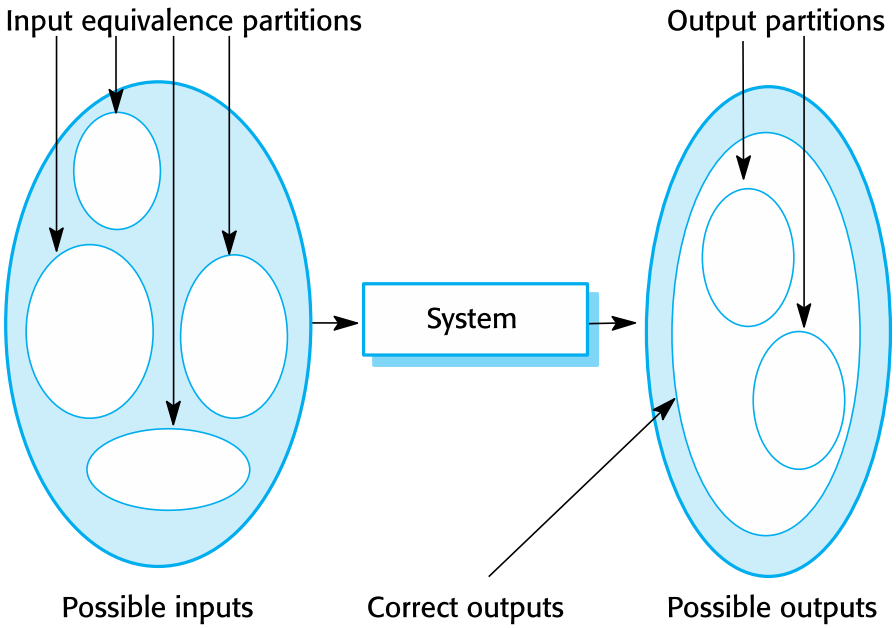
- Validation test: when used as expected, the component does what it is supposed to do.
  - The test cases reflect normal operation of a program, should show that the component works as expected.
- Defect test: If there are defects in the component, these should be revealed by test cases.
  - The test case should be based on common problems, use abnormal inputs to check that these are properly processed and do not crash the component.

Testing strategies

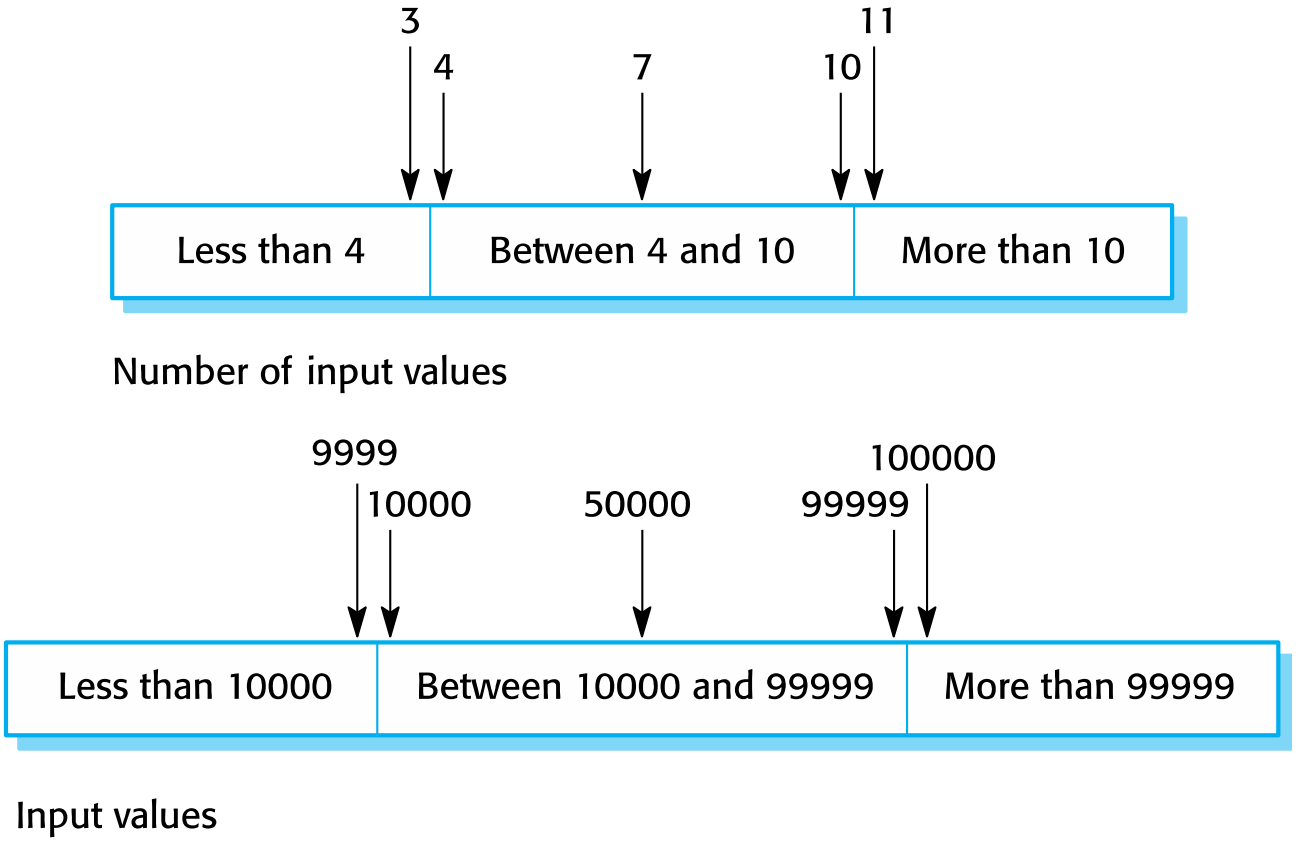
- Partition testing, where you identify groups of inputs that have common characteristics and should be processed in the same way.
  - You should choose tests from within each of these groups.
- Guideline-based testing, where you use testing guidelines to choose test cases.
  - These guidelines reflect previous experience of the kinds of errors that programmers often make when developing components.

Partition testing

- Input data and output results often fall into different classes.
- Each of these classes is an equivalence partition or domain.
- The program behaves in an equivalent way for each class member.
- Test cases should be chosen from each partition.



Example: Equivalence partitions



Testing sequences guidelines(validation)

- Test software with sequences which have only a single value.
- Use sequences of different sizes in different tests.
- Derive tests so that the first, middle and last elements of the sequence are accessed.
- Test with sequences of zero length.

General testing guidelines(detect)

- ✧ Choose inputs that force the system to generate all error messages.
- ✧ Design inputs that cause input buffers to overflow.
- ✧ Repeat the same input or series of inputs numerous times.
- ✧ Force invalid outputs to be generated.
- ✧ Force computation results to be too large or too small.

Object class testing

- Complete test coverage of a class involves:
  - Testing all operations associated with an object
  - Setting and access all object attributes
  - Exercising the object in all possible states.
- Inheritance makes it more difficult to design object class tests as the information to be tested is not localised.

Example: Weather station testing

- Need to define test cases for reportWeather, calibrate, test, startup and shutdown.
- Using a state model, identify sequences of state transitions to be tested and the event sequences to cause these transitions. For example:
  - Shutdown -> Running-> Shutdown
  - Configuring-> Running-> Testing -> Transmitting -> Running
  - Running-> Collecting-> Running-> Summarizing -> Transmitting -> Running

WeatherStation
identifier
reportWeather ( ) reportStatus ( ) powerSave (instruments) remoteControl (commands) reconfigure (commands) restart (instruments) shutdown (instruments)

T2-Component testing

- Software components are often composite components that are made up of several interacting objects.
- You access the functionality of these objects through the defined component interface.
- Testing composite components should therefore focus on showing that the component interface behaves according to its specification.

Interface testing

- Objectives are to detect faults due to interface errors or invalid assumptions about interfaces.
- Interface types
  - Parameter interfaces : Data passed from one method or procedure to another.
  - Shared memory interfaces : Block of memory is shared between procedures or functions.
  - Procedural interfaces : Sub-system encapsulates a set of procedures to be called by other sub-systems.
  - Message passing interfaces :Sub-systems request services from other sub-systems.

Interface errors

- Interface misuse
  - A calling component calls another component and makes an error in its use of its interface e.g. parameters in the wrong order.
- Interface misunderstanding
  - A calling component embeds assumptions about the behaviour of the called component which are incorrect.
- Timing errors
  - The called and the calling component operate at different speeds and out-of-date information is accessed.

Interface testing guidelines

- Design tests so that parameters to a called procedure are at the extreme ends of their ranges.
- Always test pointer parameters with null pointers.
- Design tests which cause the component to fail.
- Use stress testing in message passing systems.
- In shared memory systems, vary the order in which components are activated.

T3-System testing

- System testing during development involves integrating components to create a version of the system and then testing the integrated system.
- The focus in system testing is testing the interactions between components.
- System testing checks that components are compatible, interact correctly and transfer the right data at the right time across their interfaces.

System and component testing

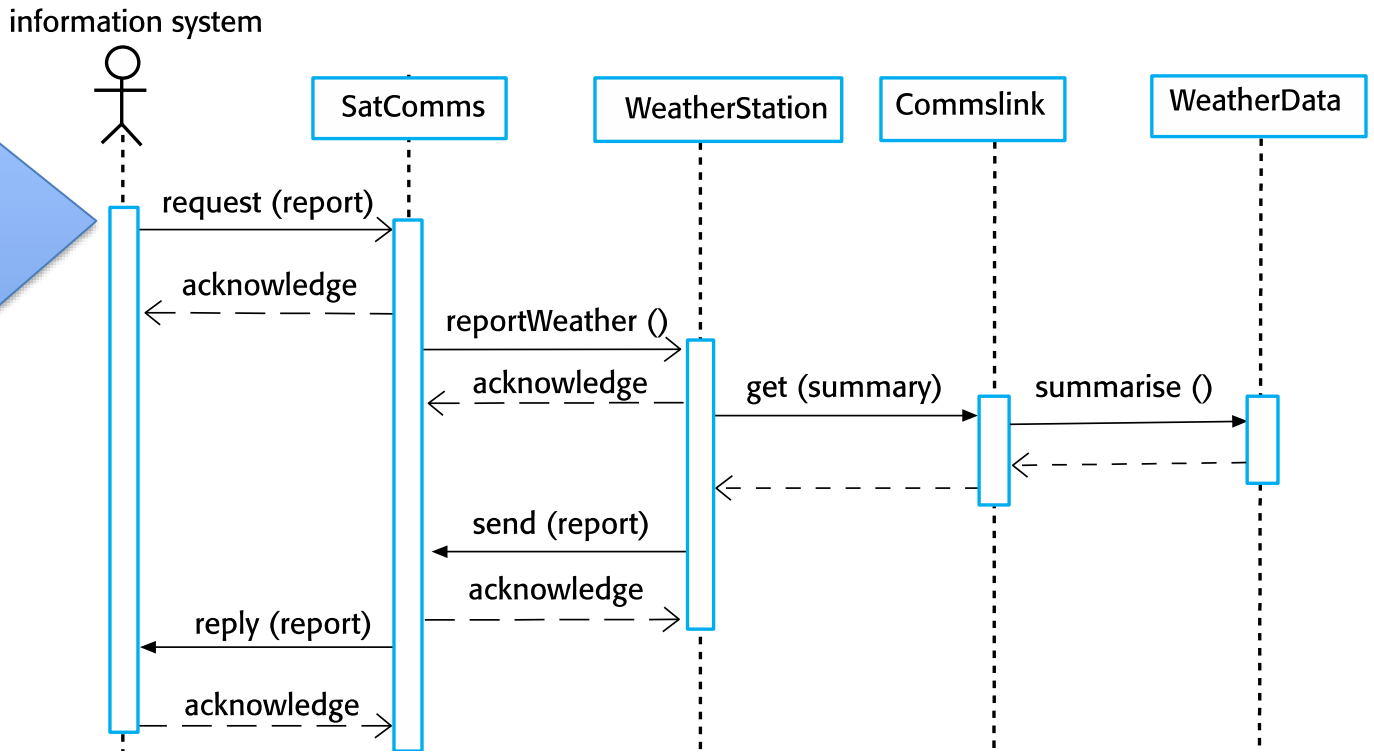
- During system testing, reusable components that have been separately developed and off-the-shelf systems may be integrated with newly developed components.
- Components developed by different team members or sub-teams may be integrated at this stage.
  - In some companies, system testing may involve a separate testing team with no involvement from designers and programmers.

Use-case testing

- The use-cases developed to identify system interactions can be used as a basis for system testing.
- Each use case usually involves several system components so testing the use case forces these interactions to occur.
- The sequence diagrams associated with the use case documents the components and interactions that are being tested.

Example: Collect weather data sequence chart

An input of a request for a report should have an associated acknowledge ment. A report should ultimately be returned from the request.



Testing policies

- Exhaustive system testing is impossible so testing policies which define the required system test coverage may be developed.
- Examples of testing policies:
  - All system functions that are accessed through menus should be tested.
  - Combinations of functions (e.g. text formatting) that are accessed through the same menu must be tested.
  - Where user input is provided, all functions must be tested with both correct and incorrect input.

T4-Release testing

- Release testing is the process of testing a particular release of a system that is intended for use outside of the development team.
- The primary goal of the release testing process is to convince the supplier of the system that it is good enough for use.
  - functionality, performance and dependability, and that it does not fail during normal use.
- Release testing is usually a black-box testing process where tests are only derived from the system specification.

Release testing and system testing

- ✧ Release testing is a form of system testing.
- ✧ Important differences:
  - A separate team that has not been involved in the system development, should be responsible for release testing.
  - System testing by the development team should focus on discovering bugs in the system (defect testing). The objective of release testing is to check that the system meets its requirements and is good enough for external use (validation testing).

Example：A usage scenario for the Mentcare system（A）

George is a nurse who specializes in mental healthcare. One of his responsibilities is to visit patients at home to check that their treatment is effective and that they are not suffering from medication side effects.

On a day for home visits, George logs into the Mentcare system and uses it to print his schedule of home visits for that day, along with summary information about the patients to be visited. He requests that the records for these patients be downloaded to his laptop. He is prompted for his key phrase to encrypt the records on the laptop.

One of the patients that he visits is Jim, who is being treated with medication for depression. Jim feels that the medication is helping him but believes that it has the side effect of keeping him awake at night. George looks up Jim's record and is prompted for his key phrase to decrypt the record. He checks the drug prescribed and queries its side effects. Sleeplessness is a known side effect so he notes the problem in Jim's record and suggests that he visits the clinic to have his medication changed. Jim agrees so George enters a prompt to call him when he gets back to the clinic to make an appointment with a physician. George ends the consultation and the system re-encrypts Jim's record.

After, finishing his consultations, George returns to the clinic and uploads the records of patients visited to the database. The system generates a call list for George of those patients who He has to contact for follow-up information and make clinic appointments.

Example：Features tested for the Mentcare system（B）

- ✧ Authentication by logging on to the system.
- ✧ Downloading and uploading of specified patient records to a laptop.
- ✧ Home visit scheduling.
- ✧ Encryption and decryption of patient records on a mobile device.
- ✧ Record retrieval and modification.
- ✧ Links with the drugs database that maintains side-effect information.
- ✧ The system for call prompting.

Performance testing

- Part of release testing may involve testing the emergent properties of a system, such as performance and reliability.
- Performance tests usually involve planning a series of tests where the load is steadily increased until the system performance becomes unacceptable.
- Stress testing is a form of performance testing where the system is deliberately overloaded to test its failure behaviour.

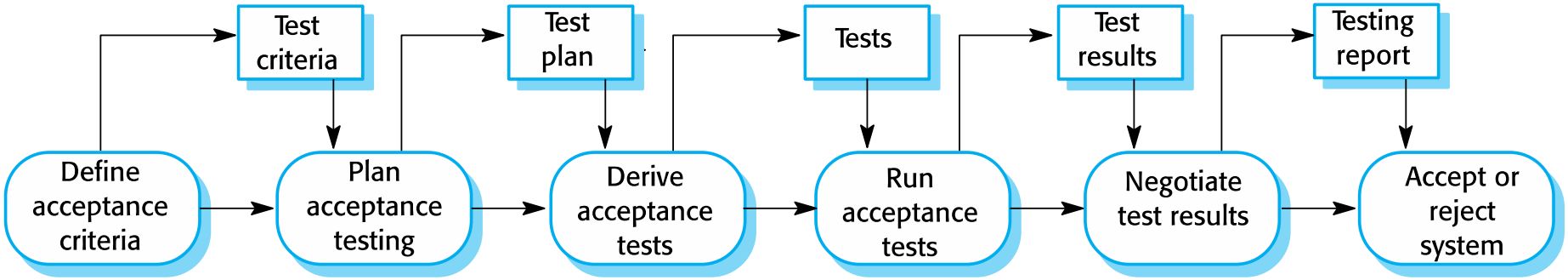
T5-User testing

- User or customer testing is a stage in the testing process in which users or customers provide input and advice on system testing.
- User testing is essential, even when comprehensive system and release testing have been carried out.
  - The influences from the user's working environment have a major effect on the reliability, performance, usability and robustness of a system.

Types of user testing

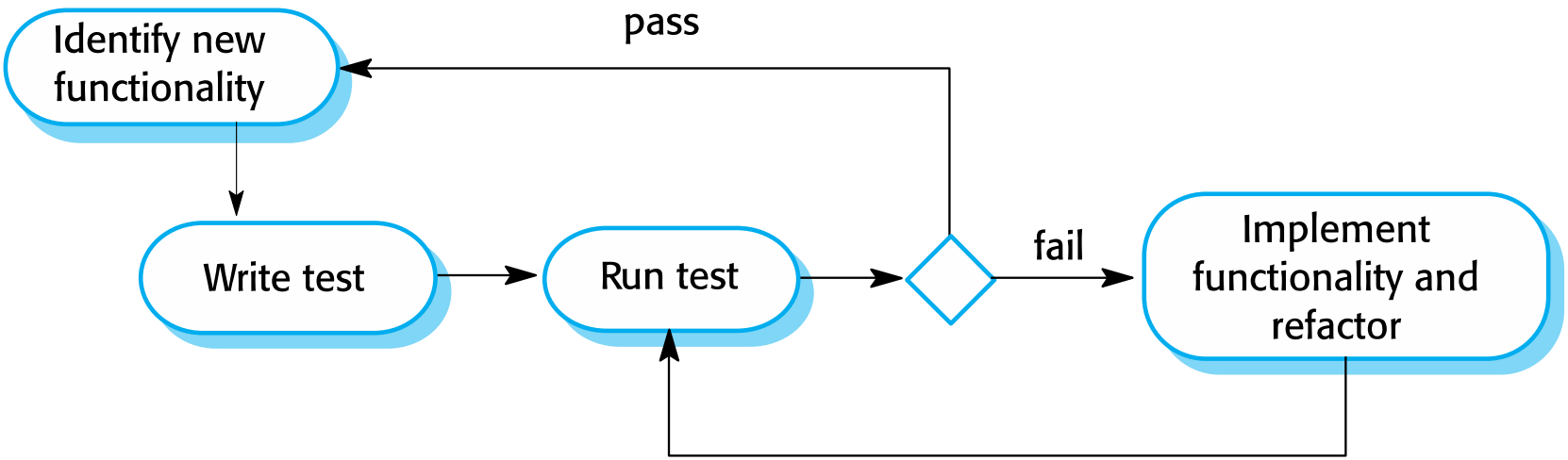
- ✧ Alpha testing
  - Users of the software work with the development team to test the software at the developer's site.
- ✧ Beta testing
  - A release of the software is made available to users to allow them to experiment and to raise problems that they discover with the system developers.
- ✧ Acceptance testing（Primarily for custom systems）
  - Customers test a system to decide whether or not it is ready to be accepted and deployed in the customer environment.

The acceptance testing process



Test-driven development

- Test-driven development (TDD) is an approach in which you inter-leave testing and code development.



- TDD was introduced as part of agile methods such as Extreme Programming. However, it can also be used in plan-driven development processes.

Benefits of test-driven development

- ✧ Code coverage
  - Every code segment has at least one associated test.
- ✧ Regression testing
  - Regression testing is testing the system to check that changes have not ‘broken’ previously working code.
  - Regression test suite is developed incrementally as a program is developed.
- ✧ Simplified debugging
  - When a test fails, it should be obvious where the problem lies.
- ✧ System documentation
  - The tests themselves are a form of documentation that describe what the code should be doing.

Benefits of test-driven development

- Code coverage
  - Every code segment has at least one associated test.
- Regression testing
  - Regression testing is testing the system to check that changes have not ‘broken’ previously working code.
  - Regression test suite is developed incrementally as a program is developed.
- Simplified debugging
  - When a test fails, it should be obvious where the problem lies.
- System documentation
  - The tests themselves are a form of documentation that describe what the code should be doing.

Automated testing

- Whenever possible, unit testing should be automated so that tests are run and checked without manual intervention.
- In automated unit testing, you make use of a test automation framework (such as JUnit) to write and run your program tests.
- Unit testing frameworks provide generic test classes that you extend to create specific test cases. They can then run all of the tests that you have implemented and report, often through some GUI, on the success of otherwise of the tests.

lecture8 : Testing and Quality

- **Reading for this week:**
  - ' Agile Software Testing in a Large-Scale Project' by Bojana Koteska, Anastas Mishev
  - Chapter 8: Software testing from the coursebook (Software Engineering by Ian Sommerville)
  - Chapter 24: Quality management from the coursebook (Software Engineering by Ian Sommerville)
- **Assignments for this week:**
  - Quiz 8: 2 attempts, 30 minutes time
  - Essay6: Software Quality and Testing.