Въведение в управление на качеството. Верификация и валидация на софтуерни системи.

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

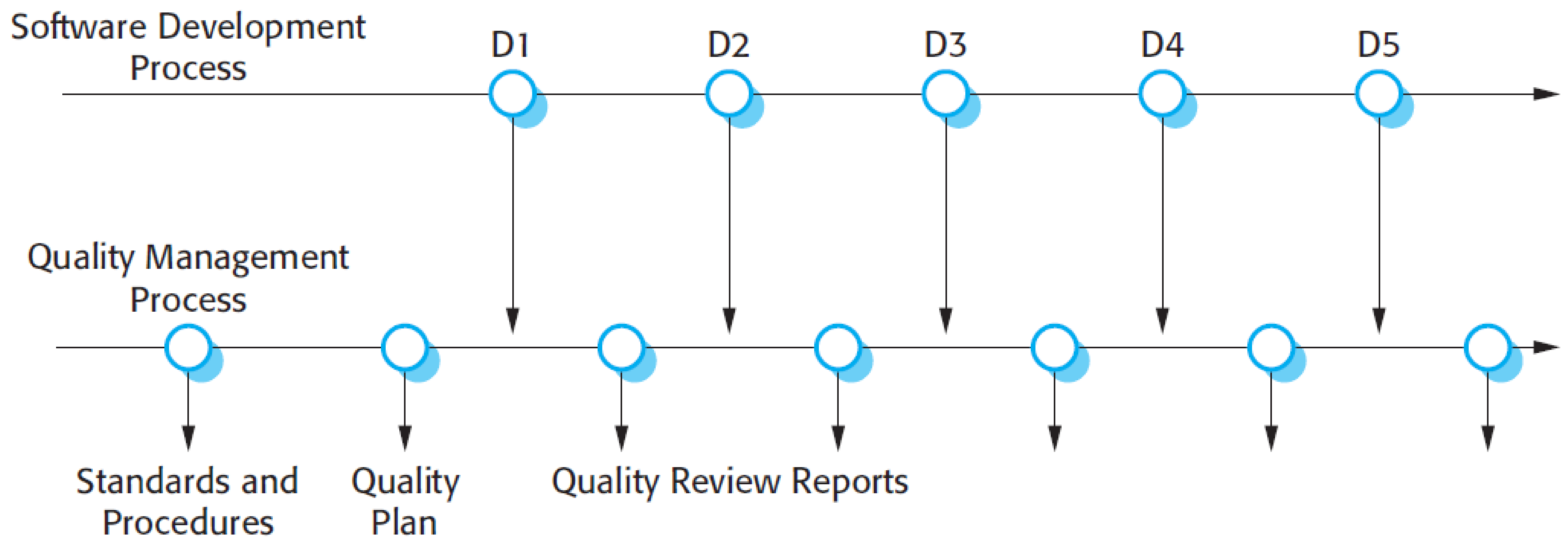


Image from: Sommerville, I. Engineering Software Products: An Introduction to Modern Software Engineering 1st Edition, Published by Pearson, ISBN: 978-0135210642, (2019)

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**.

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 or 'The software is suitable for its purpose?'

Software fitness for purpose

- Have programming and documentation standards been followed in the development process?
- 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?

Software quality attributes

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

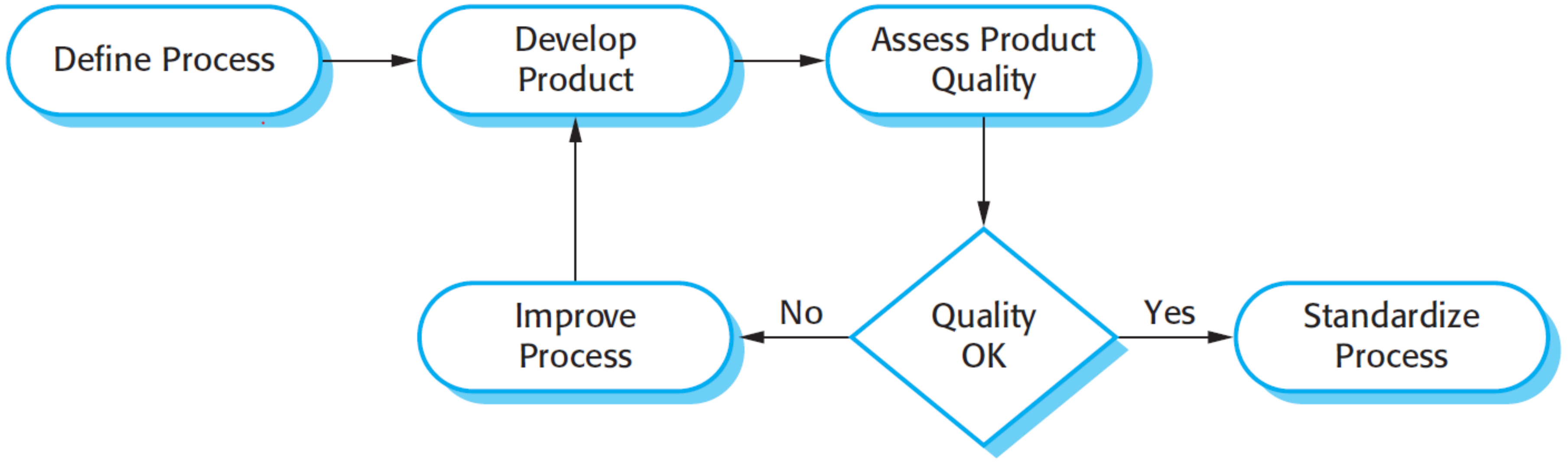


Image from: Sommerville, I. Engineering Software Products: An Introduction to Modern Software Engineering 1st Edition, Published by Pearson, ISBN: 978-0135210642, (2019)

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.**
- **Product standards** define characteristics that all software components should exhibit e.g. a common programming style.
- **Process standards** define how the software process should be enacted.

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	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

Software testing

Software testing

- **Software testing** is a process in which you execute your program using data that simulates user inputs.
- You observe its behaviour to see whether or not your program is doing what it is supposed to do.
  - *Tests pass if the behaviour is what you expect.*
  - *Tests fail if the behaviour differs from that expected.*
  - *If your program does what you expect, this shows that for the inputs used, the program behaves correctly.*
- If these inputs are representative of a larger set of inputs, you can infer that the program will behave correctly for all members of this larger input set.

Program bugs

- If the behaviour of the program does not match the behaviour that you expect, then this means that there are bugs in your program that need to be fixed.
- There are two causes of program bugs:
  - ***Programming errors*** You have accidentally included faults in your program code. For example, a common programming error is an 'off-by-1' error where you make a mistake with the upper bound of a sequence and fail to process the last element in that sequence.
  - ***Understanding errors*** You have misunderstood or have been unaware of some of the details of what the program is supposed to do. For example, if your program processes data from a file, you may not be aware that some of this data is in the wrong format, so your program doesn't include code to handle this.

Types of testing

- ***Functional testing***
  Test the functionality of the overall system. The goals of functional testing are to discover as many bugs as possible in the implementation of the system and to provide convincing evidence that the system is fit for its intended purpose.
- ***User testing***
  Test that the software product is useful to and usable by end-users. You need to show that the features of the system help users do what they want to do with the software. You should also show that users understand how to access the software's features and can use these features effectively.
- ***Performance and load testing***
  Test that the software works quickly and can handle the expected load placed on the system by its users. You need to show that the response and processing time of your system is acceptable to end-users. You also need to demonstrate that your system can handle different loads and scales gracefully as the load on the software increases.
- ***Security testing***
  Test that the software maintains its integrity and can protect user information from theft and damage.

Functional testing

- **Functional testing** involves developing a large set of program tests so that, ideally, all of a program's code is executed at least once.
- The number of tests needed obviously depends on the size and the functionality of the application.
- For a business-focused web application, you may have to develop thousands of tests to convince yourself that your product is ready for release to customers.
- **Functional testing is a staged activity in which you initially test individual units of code.** You integrate code units with other units to create larger units then do more testing.
- The process continues until you have created a complete system ready for release.

Functional testing

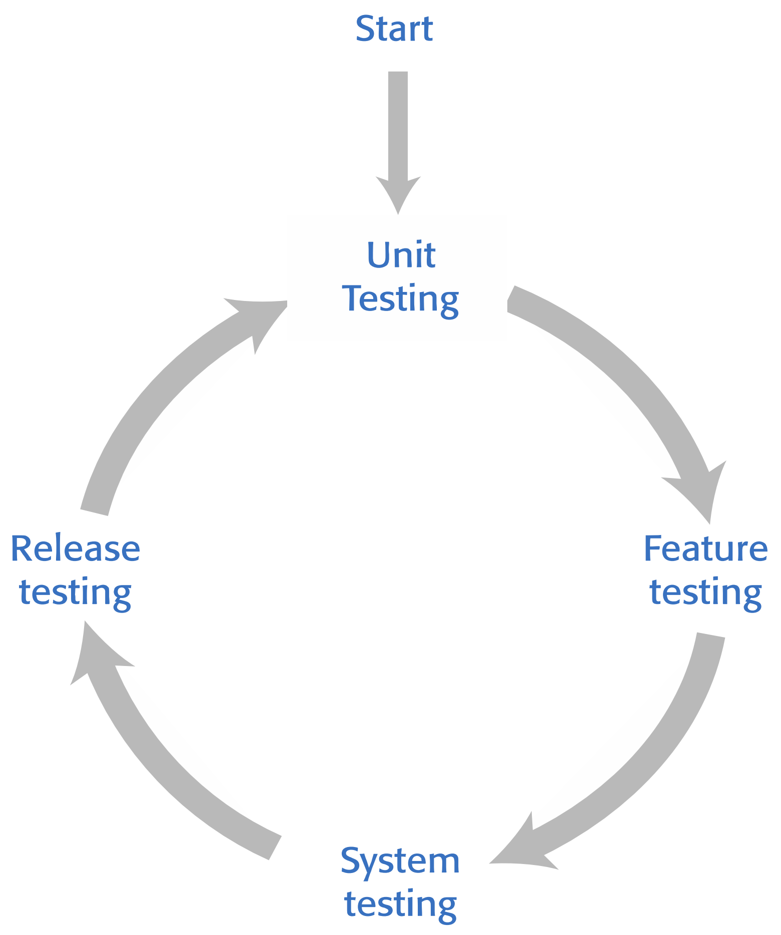


Image from: Sommerville, I. Engineering Software Products: An Introduction to Modern Software Engineering 1st Edition, Published by Pearson, ISBN: 978-0135210642, (2019)

Functional testing processes

- ***Unit testing***
  The aim of unit testing is to test program units in isolation. Tests should be designed to execute all of the code in a unit at least once. Individual code units are tested by the programmer as they are developed.
- ***Feature testing***
  Code units are integrated to create features. Feature tests should test all aspects of a feature. All of the programmers who contribute code units to a feature should be involved in its testing.
- ***System testing***
  Code units are integrated to create a working (perhaps incomplete) version of a system. The aim of system testing is to check that there are no unexpected interactions between the features in the system. System testing may also involve checking the responsiveness, reliability and security of the system. In large companies, a dedicated testing team may be responsible for system testing. In small companies, this is impractical, so product developers are also involved in system testing.
- ***Release testing***
  The system is packaged for release to customers and the release is tested to check that it operates as expected. The software may be released as a cloud service or as a download to be installed on a customer's computer or mobile device. If DevOps is used, then the development team are responsible for release testing otherwise a separate team has that responsibility.

Unit testing

- As you develop a code unit, you should also develop tests for that code.
- A code unit is anything that has a clearly defined responsibility. It is usually a function or class method but could be a module that includes a small number of other functions.
- Unit testing is based on a simple general principle:
  - If a program unit behaves as expected for a set of inputs that have some shared characteristics, it will behave in the same way for a larger set whose members share these characteristics.

Feature testing

- Features have to be tested to show that the functionality is implemented as expected and that the functionality meets the real needs of users.
  - For example, if your product has a feature that allows users to login using their Google account, then you have to check that this registers the user correctly and informs them of what information will be shared with Google.
  - You may want to check that it gives users the option to sign up for email information about your product.
- Normally, a feature that does several things is implemented by multiple, interacting, program units.
- These units may be implemented by different developers and all of these developers should be involved in the feature testing process.

Types of feature test

- **Interaction tests**
  - These test the interactions between the units that implement the feature. The developers of the units that are combined to make up the feature may have different understandings of what is required of that feature.
  - These misunderstandings will not show up in unit tests but may only come to light when the units are integrated.
  - The integration may also reveal bugs in program units, which were not exposed by unit testing.
- **Usefulness tests**
  - These test that the feature implements what users are likely to want.
  - For example, the developers of a login with Google feature may have implemented an opt-out default on registration so that users receive all emails from a company. They must expressly choose what type of emails that they don't want.
  - What might be preferred is an opt-in default so that users choose what types of email they do want to receive.

System and release testing

- System testing involves testing the system as a whole, rather than the individual system features.
- **System testing should focus on four things:**
  - Testing to discover if there are unexpected and unwanted interactions between the features in a system.
  - Testing to discover if the system features work together effectively to support what users really want to do with the system.
  - Testing the system to make sure it operates in the expected way in the different environments where it will be used.
  - Testing the responsiveness, throughput, security and other quality attributes of the system.

Scenario-based testing

- The best way to **systematically test a system is to start with a set of scenarios that describe possible uses of the system** and then work through these scenarios each time a new version of the system is created.
- Using the **scenario**, you **identify a set of end-to-end pathways that users might follow when using the system.**
- An **end-to-end pathway** is **a sequence of actions from starting to use the system for the task, through to completion of the task**.

Release testing

- Release testing is a type of system testing where a system that's intended for release to customers is tested.
- **The fundamental differences between release testing and system testing are:**
  - Release testing tests the system in its real operational environment rather than in a test environment. Problems commonly arise with real user data, which is sometimes more complex and less reliable than test data.
  - The aim of release testing is to decide if the system is good enough to release, not to detect bugs in the system. Therefore, some tests that 'fail' may be ignored if these have minimal consequences for most users.
- Preparing a system for release involves packaging that system for deployment (e.g. in a container if it is a cloud service) and installing software and libraries that are used by your product. You must define configuration parameters such as the name of a root directory, the database size limit per user and so on.

Test automation

- Automated testing is based on the idea that tests should be executable.
- An executable test includes the input data to the unit that is being tested, the expected result and a check that the unit returns the expected result.
- You run the test and the test passes if the unit returns the expected result.
- Normally, you should develop hundreds or thousands of executable tests for a software product.

Automated testing

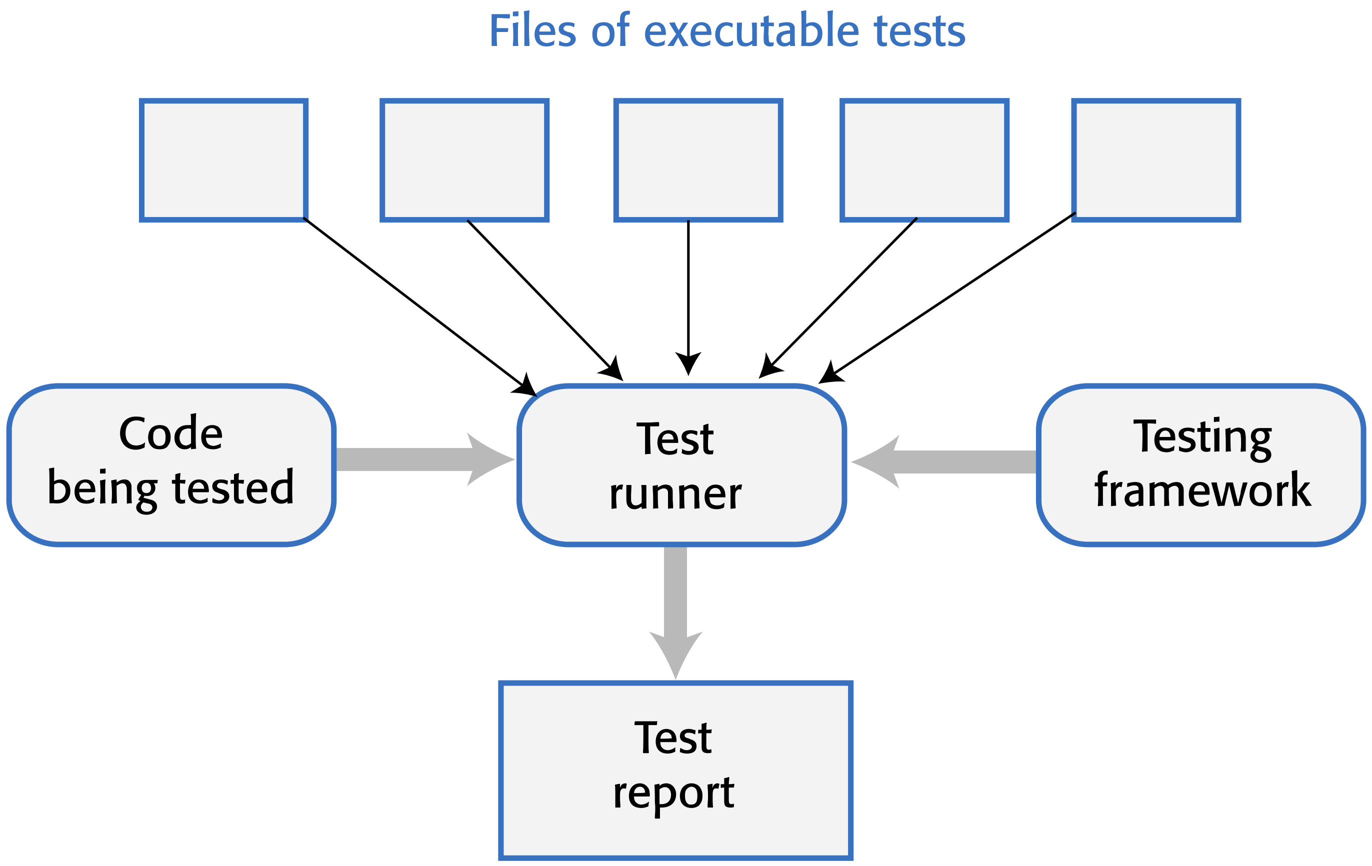


Image from: Sommerville, I. Engineering Software Products: An Introduction to Modern Software Engineering 1st Edition, Published by Pearson, ISBN: 978-0135210642, (2019)

The test pyramid

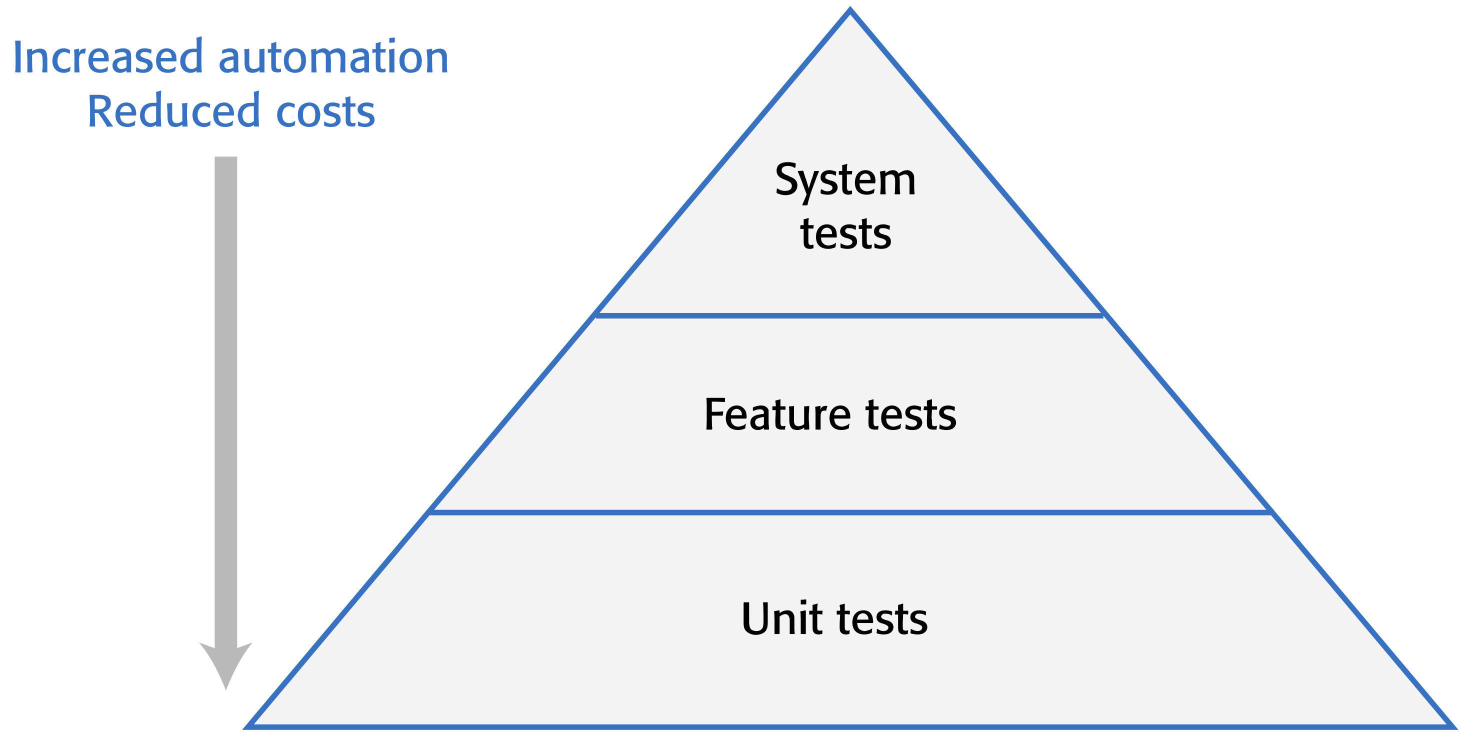
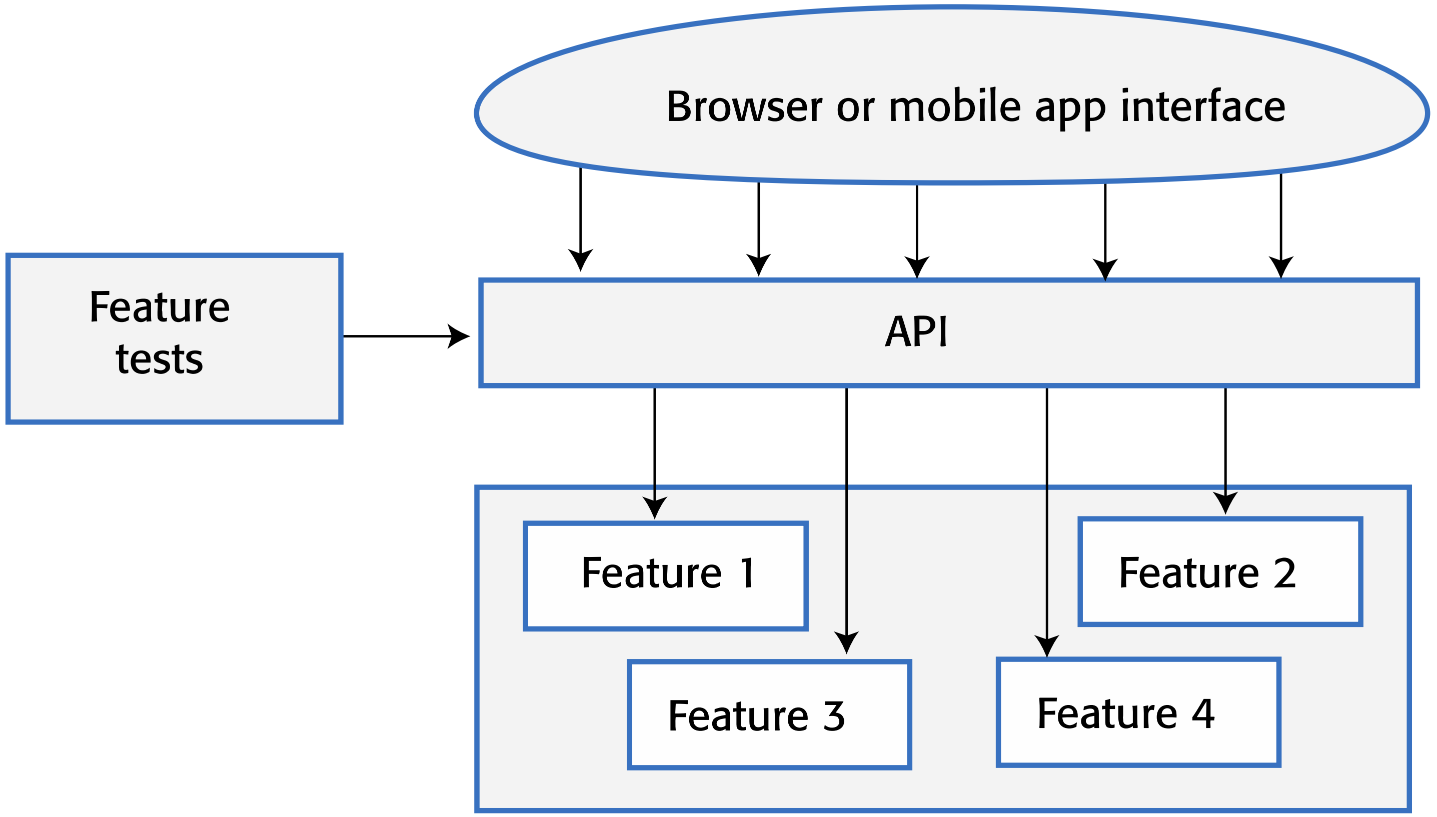


Image from: Sommerville, I. Engineering Software Products: An Introduction to Modern Software Engineering 1st Edition, Published by Pearson, ISBN: 978-0135210642, (2019)

Automated feature testing

- Generally, users access features through the product's graphical user interface (GUI).
- However, GUI-based testing is expensive to automate so it is best to design your product so that its features can be directly accessed through an API and not just from the user interface.
- The feature tests can then access features directly through the API without the need for direct user interaction through the system's GUI.
- Accessing features through an API has the additional benefit that it is possible to re-implement the GUI without changing the functional components of the software.

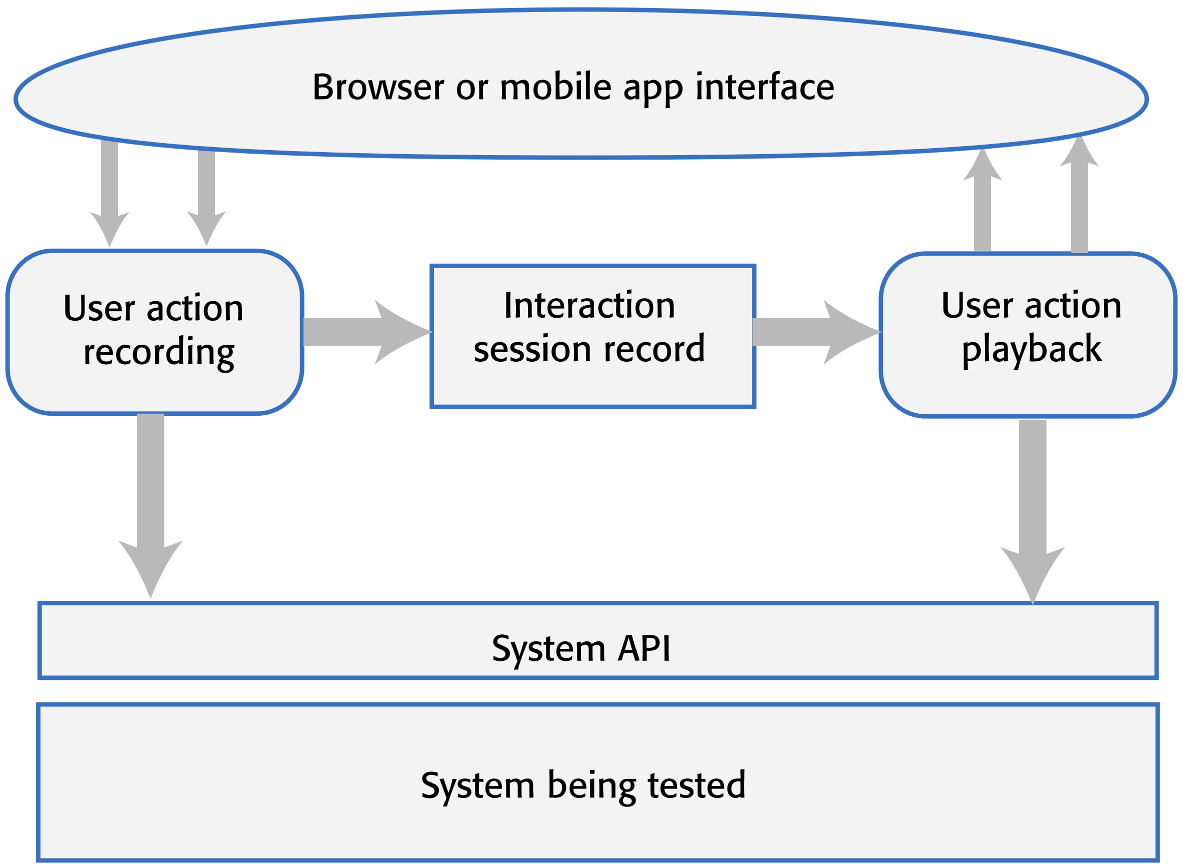
Feature editing through an API



System testing

- System testing, which should follow feature testing, involves testing the system as a surrogate user.
- As a system tester, you go through a process of selecting items from menus, making screen selections, inputting information from the keyboard and so on.
- You are looking for interactions between features that cause problems, sequences of actions that lead to system crashes and so on.
- Manual system testing, when testers have to repeat sequences of actions, is boring and error-prone. In some cases, the timing of actions is important and is practically impossible to repeat consistently.
  - To avoid these problems, testing tools have been developed that can record a series of actions and automatically replay these when a system is retested

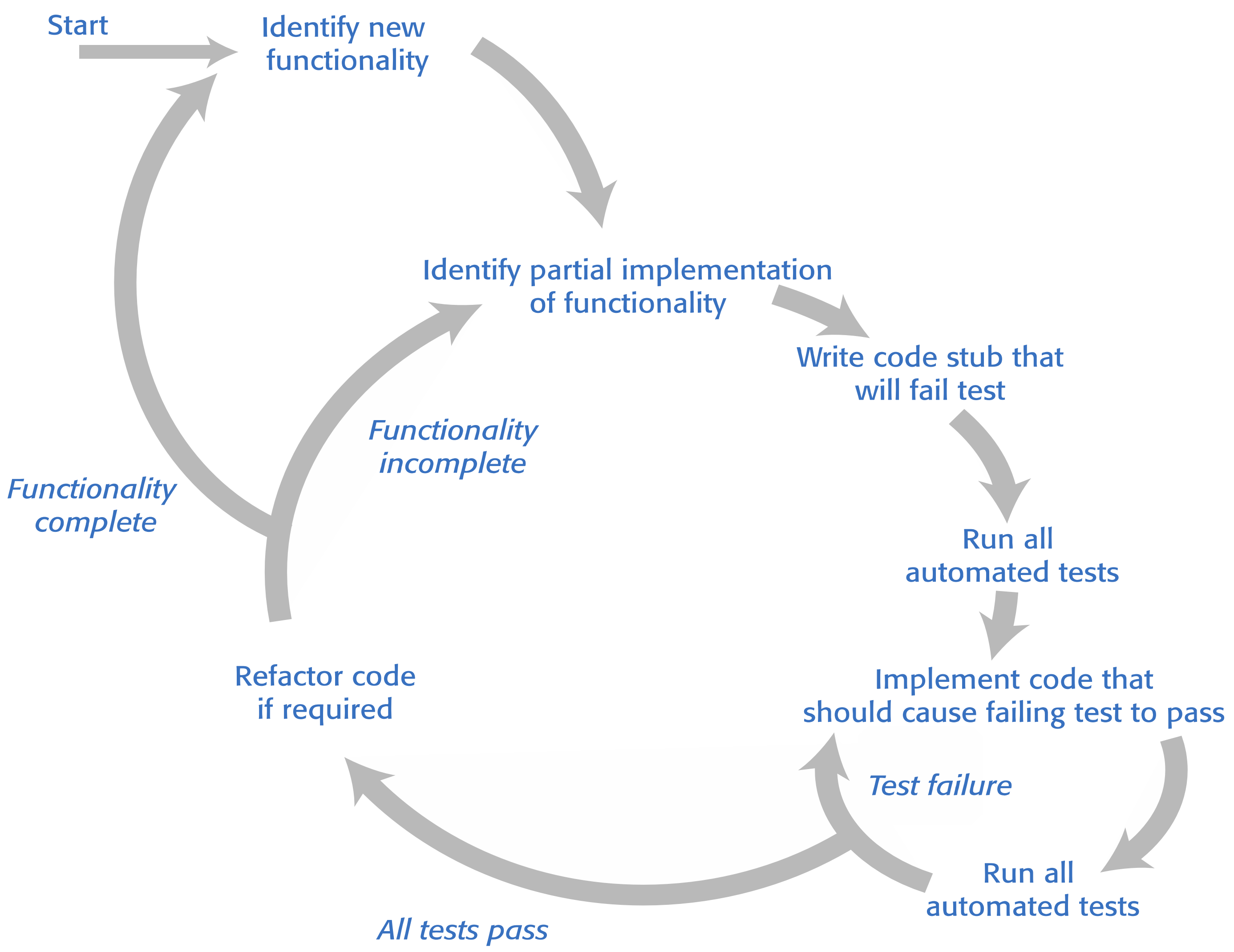
Interaction recording and playback



Test-driven development

- Test-driven development (TDD) is an approach to program development that is based around the general idea that **you should write an executable test or tests for code that you are writing before you write the code**.
- It was introduced by early users of the Extreme Programming agile method, but it can be used with any incremental development approach.
- Test-driven development works best for the development of individual program units and it is more difficult to apply to system testing.
- Even the strongest advocates of TDD accept that it is challenging to use this approach when you are developing and testing systems with graphical user interfaces.

Test-driven development


Start
Identify new functionality
Identify partial implementation of functionality
Write code stub that will fail test
Run all automated tests
Implement code that should cause failing test to pass
Test failure
Run all automated tests
All tests pass
Refactor code if required
Functionality incomplete
Functionality complete

Stages of test-driven development (1)

- ***Identify partial implementation***
  Break down the implementation of the functionality required into smaller mini-units. Choose one of these mini-units for implementation.
- ***Write mini-unit tests***
  Write one or more automated tests for the mini-unit that you have chosen for implementation. The mini-unit should pass these tests if it is properly implemented.
- **Write a code stub that will fail test**
  Write incomplete code that will be called to implement the mini-unit. You know this will fail.
- ***Run all existing automated tests***
  All previous tests should pass. The test for the incomplete code should fail.

Stages of test-driven development (2)

- ***Implement code that should cause the failing test to pass***
  Write code to implement the mini-unit, which should cause it to operate correctly
- ***Rerun all automated tests***
  If any tests fail, your code is probably incorrect. Keep working on it until all tests pass.
- ***Refactor code if necessary***
  If all tests pass, you can move on to implementing the next mini-unit. If you see ways of improving your code, you should do this before the next stage of implementation.

Benefits of test-driven development

- It is a systematic approach to testing in which tests are clearly linked to sections of the program code.
  - This means you can be confident that your tests cover all of the code that has been developed and that there are no untested code sections in the delivered code - this is the most significant benefit of TDD.
- The tests act as a written specification for the program code. In principle at least, it should be possible to understand what the program does by reading the tests.
- Debugging is simplified because, when a program failure is observed, you can immediately link this to the last increment of code that you added to the system.
- It is argued that TDD leads to simpler code as programmers only write code that's necessary to pass tests. They don't over-engineer their code with complex features that aren't needed.

Security testing

- Security testing aims to find vulnerabilities that may be exploited by an attacker and to provide convincing evidence that the system is sufficiently secure.
- The tests should demonstrate that the system can resist attacks on its availability, attacks that try to inject malware and attacks that try to corrupt or steal users' data and identity.
- Comprehensive security testing requires specialist knowledge of software vulnerabilities and approaches to testing that can find these vulnerabilities.

Risk-based security testing

- A risk-based approach to security testing involves identifying common risks and developing tests to demonstrate that the system protects itself from these risks.
- You may also use automated tools that scan your system to check for known vulnerabilities, such as unused HTTP ports being left open.
- Based on the risks that have been identified, you then design tests and checks to see if the system is vulnerable.
- It may be possible to construct automated tests for some of these checks, but others inevitably involve manual checking of the system's behaviour and its files.

Examples of security risks

- Unauthorized attacker gains access to a system using authorized credentials
- Authorized individual accesses resources that are forbidden to them
- Authentication system fails to detect unauthorized attacker
- Attacker gains access to database using SQL poisoning attack
- Improper management of HTTP session
- HTTP session cookies revealed to attacker
- Confidential data are unencrypted
- Encryption keys are leaked to potential attackers

Code reviews

- Code reviews involve one or more people examining the code to check for errors and anomalies and discussing issues with the developer.
- If problems are identified, it is the developer's responsibility to change the code to fix the problems.
- Code reviews complement testing. They are effective in finding bugs that arise through misunderstandings and bugs that may only arise when unusual sequences of code are executed.
- Many software companies insist that all code has to go through a process of code review before it is integrated into the product codebase.

Code reviews

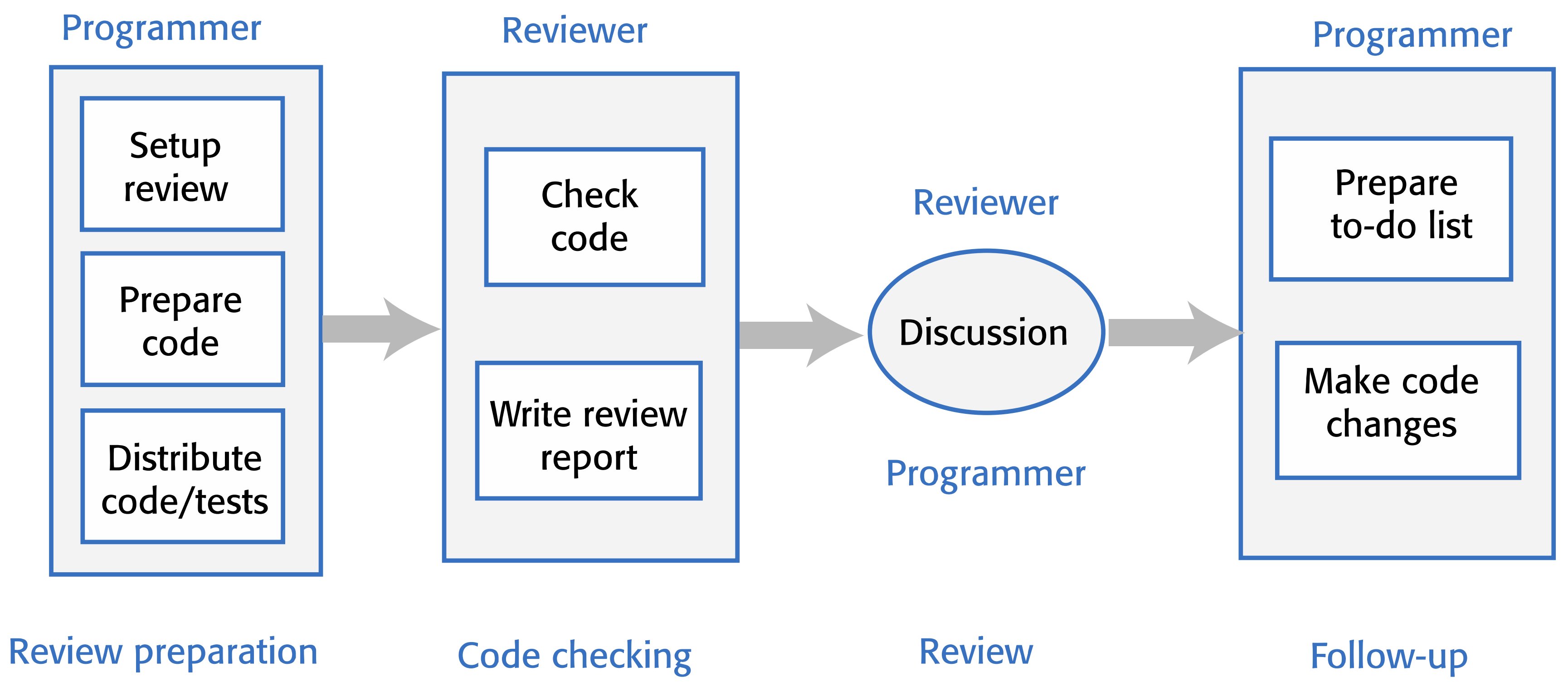


Image from: Sommerville, I. Engineering Software Products: An Introduction to Modern Software Engineering 1st Edition, Published by Pearson, ISBN: 978-0135210642, (2019)

Code review activities (1)

- ***Setup review***
  The programmer contacts a reviewer and arranges a review date.
- ***Prepare code***
  The programmer collects the code and tests for review and annotates them with information for the reviewer about the intended purpose of the code and tests.
- ***Distribute code/tests***
  The programmer sends code and tests to the reviewer.
- ***Check code***
  The reviewer systematically checks the code and tests against their understanding of what they are supposed to do.
- ***Write review report***
  The reviewer annotates the code and tests with a report of the issues to be discussed at the review meeting.

Code review activities (2)

- ***Discussion***
  The reviewer and programmer discuss the issues and agree on the actions to resolve these.
- ***Make to-do list***
  The programmer documents the outcome of the review as a to-do list and shares this with the reviewer.
- ***Make code changes***
  The programmer modifies their code and tests to address the issues raised in the review.

Conclusion: Key points 1

- 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 and so on.
- SQM includes defining standards for processes and products and establishing processes to check that these standards have been followed.
- Software standards are important for quality assurance as they represent an identification of 'best practice'.
- The aim of program testing is to find bugs and to show that a program does what its developers expect it to do.
- Four types of testing that are relevant to software products are functional testing, user testing, load and performance testing and security testing.

Conclusion: Key points 2

- Unit testing involves testing program units such as functions or class methods that have a single responsibility. Feature testing focuses on testing individual system features. System testing tests the system as a whole to check for unwanted interactions between features and between the system and its environment.
- User stories may be used as a basis for deriving feature tests.
- Test automation is based on the idea that tests should be executable. You develop a set of executable tests and run these each time you make a change to a system.

Conclusion: Key points 3

- Test-driven development is an approach to development where executable tests are written before the code. Code is then developed to pass the tests.
- A disadvantage of test-driven development is that programmers focus on the detail of passing tests rather than considering the broader structure of their code and algorithms used.
- Security testing may be risk driven where a list of security risks is used to identify tests that may identify system vulnerabilities.
- Code reviews are an effective supplement to testing. They involve people checking the code to comment on the code quality and to look for bugs.

References

- *Sommerville, I. Software Engineering. 9th edition, Published by Pearson Education, ISBN: 978-0-13-703515-1 (2011)*
- *Sommerville, I. Software Engineering. 10th edition, Published by Pearson Education, ISBN: 978-1-292-09613-1 (2016)*
- *Pressman, R., Maxim, B. Software Engineering: A Practitioner's Approach. 9th edition, Published by McGraw-Hill Education, ISBN: 9781260548006, (2019)*
- *Sommerville, I. Engineering Software Products: An Introduction to Modern Software Engineering 1st Edition, Published by Pearson, ISBN: 978-0135210642, (2019)*