# Software Quality Assurance and Testing

Prof. Casper Lassenius

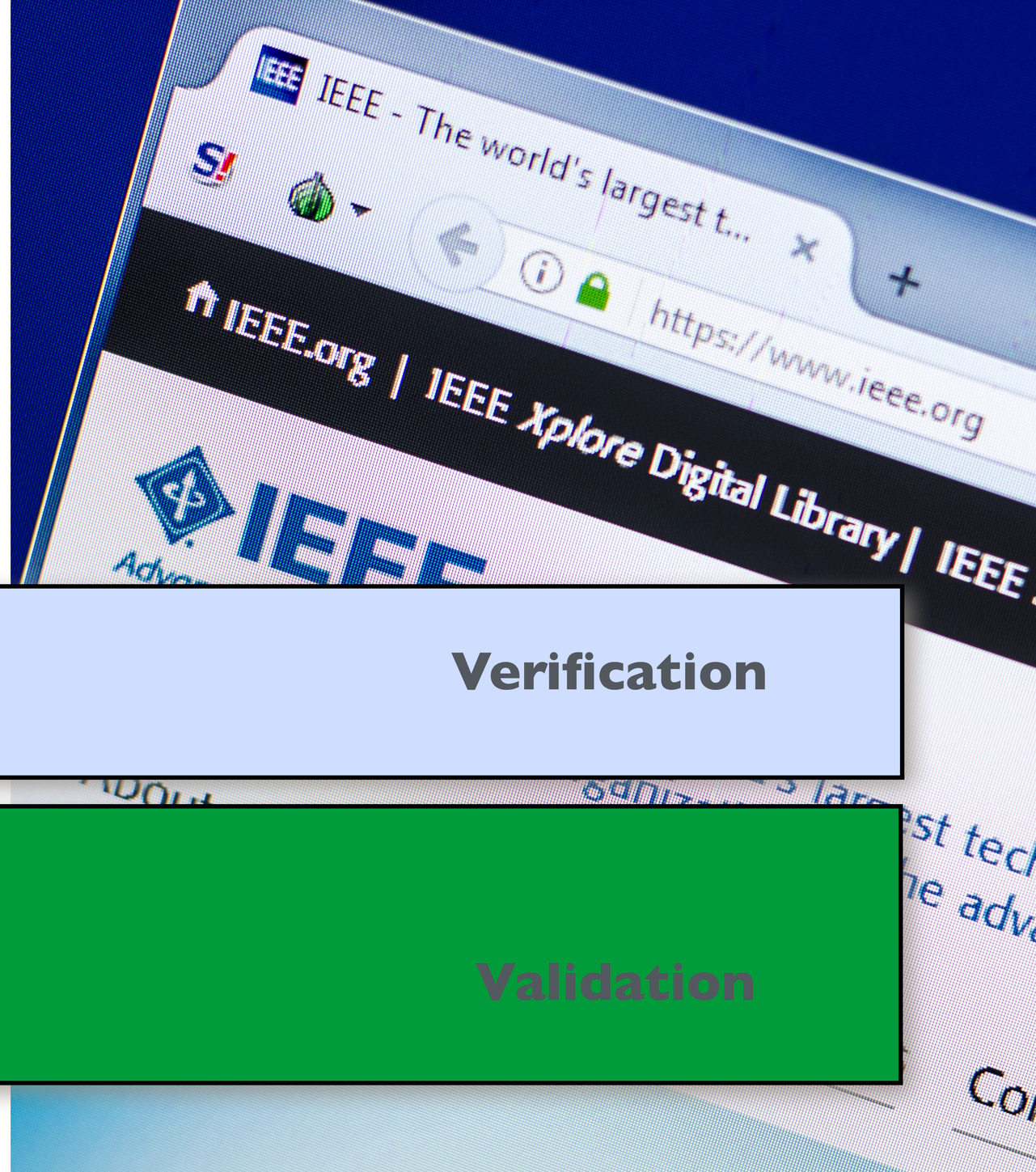
# What is Software Quality?

# 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

- Standard Glossary of Software Engineering Terminology [IEEE610.12]:




- The degree to which a system, component or process meets specified requirements

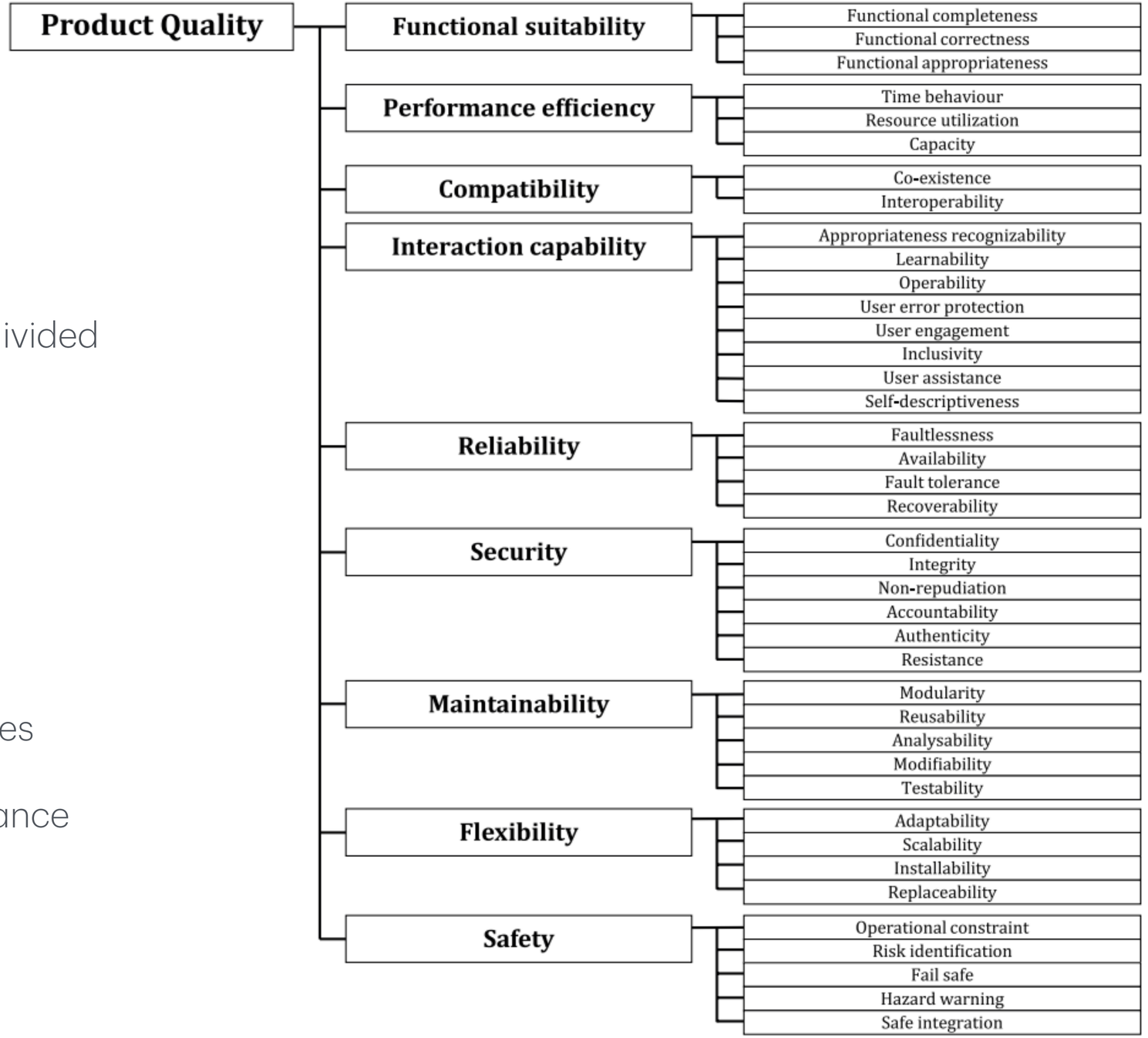
**Verification**

- The degree to which a system, component, or process meets customer or user needs or expectations

**Validation**

# ISO 25010 Product Quality Characteristics

- Nine main quality characteristics, each divided into sub-characteristics
- Uses
  - Eliciting and defining requirements
  - Validating the requirements definition
  - Identifying design and testing objectives
  - Identifying quality control and acceptance criteria
  - Establishing quality measures



| User needs | Primary user | Secondary users | | Indirect user |
|---|---|---|---|---|
| | | Content provider | Maintainer | |
| | Interacting | Interacting | Maintaining or porting | Using output |
| Reliability | How reliable does the system need to be when the user uses it to perform their task? | How reliable does updating the system with new content need to be? | How reliable does maintaining or porting the system need to be? | How reliable does the output from the system need to be? |
| Security | How secure does the system need to be when the user uses it to perform their task? | How secure does the system need to be after the content provider updates it? | How secure does the system need to be after maintenance changes are made or when it is ported? | How secure does the output from the system need to be? |
| Learnability | To what extent does learning to use the system need to be effective, efficient, risk free and satisfying? | To what extent does learning to provide content need to be effective, efficient, risk free and satisfying? | To what extent does learning to maintain or port the system need to be effective, efficient, risk free and satisfying? | To what extent does learning to use the output from the system need to be effective, efficient, risk free and satisfying? |
| Inclusivity and user assistance | To what extent does the system need to be effective, efficient, risk free and satisfying to use for people with disabilities? | To what extent does providing content for the system need to be effective, efficient, risk free and satisfying for people with disabilities? | To what extent does maintaining or porting the system need to be effective, efficient, risk free and satisfying for people with disabilities? | To what extent does using the output of the system need to be effective, efficient, risk free and satisfying for people with disabilities? |

# Stakeholder Viewpoints

- All stakeholders have different viewpoints
  - Customer
  - User
  - Programmer
  - Project manager
  - Tester
- Note: even a bug-free product can be unacceptable to the user
  - User scenarios (use-cases) should be validated
  - Requirements should be validated
  - Usability needs to be considered
  - Documentation should be tested
  - Customer feedback process needs to be working

# 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.

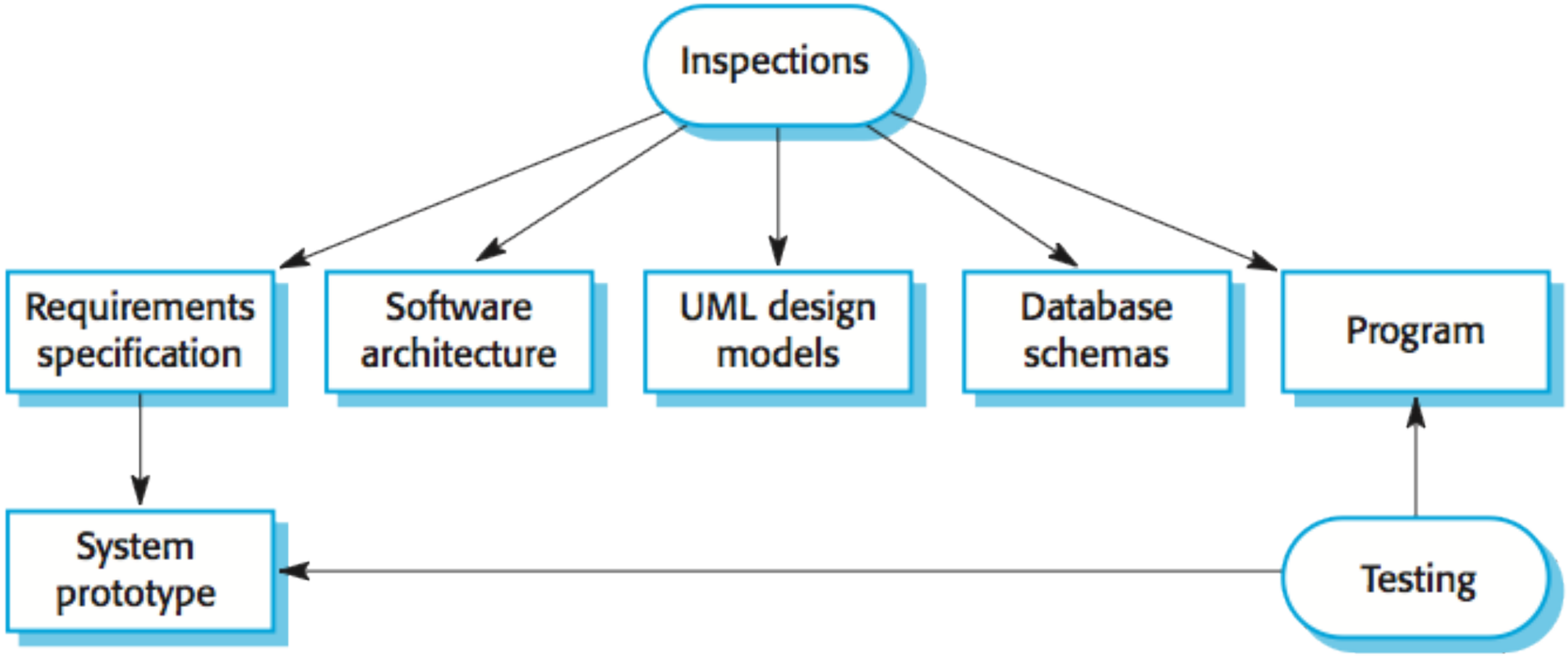
# Process and product quality

- 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.

# Ensuring Software Quality

- Understand what quality means in your context
- Build quality in using appropriate processes
- Quality assurance
  - Testing
  - Reviews / Inspections

# Inspections and testing



# 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).

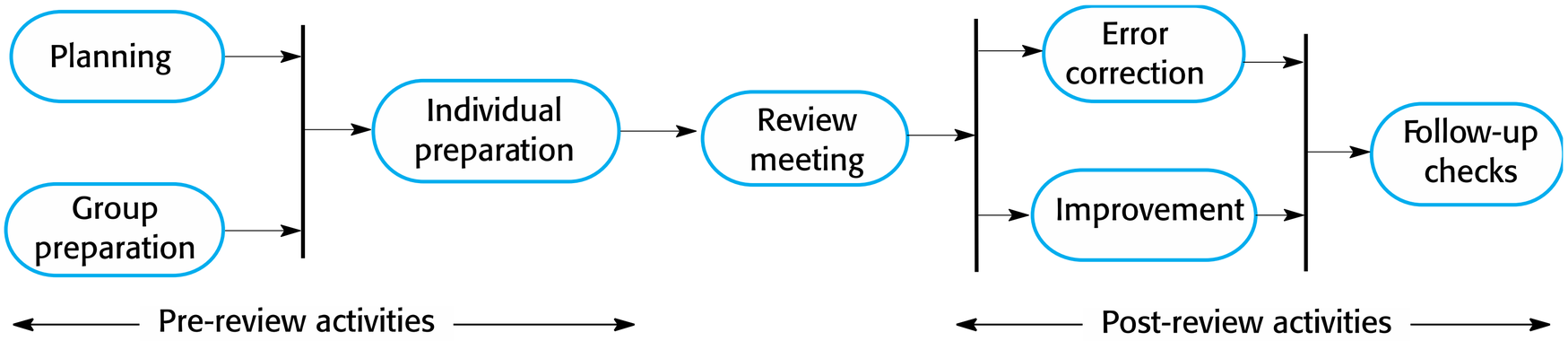
# Quality reviews

- 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.

# Inspections and testing

- 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.

# The software review process



# Reviews and agile methods

- The review process in agile software development is usually informal.
  - In Scrum, for example, 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.
- XP relies on individuals taking the initiative to improve and refactor code. Agile approaches are not usually standards-driven, so issues of standards compliance are not usually considered.

# 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.

# 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. If a program is incomplete, then you need to develop specialized test harnesses to test the parts that are available.
- 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.

# 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.

# Agile methods and inspections

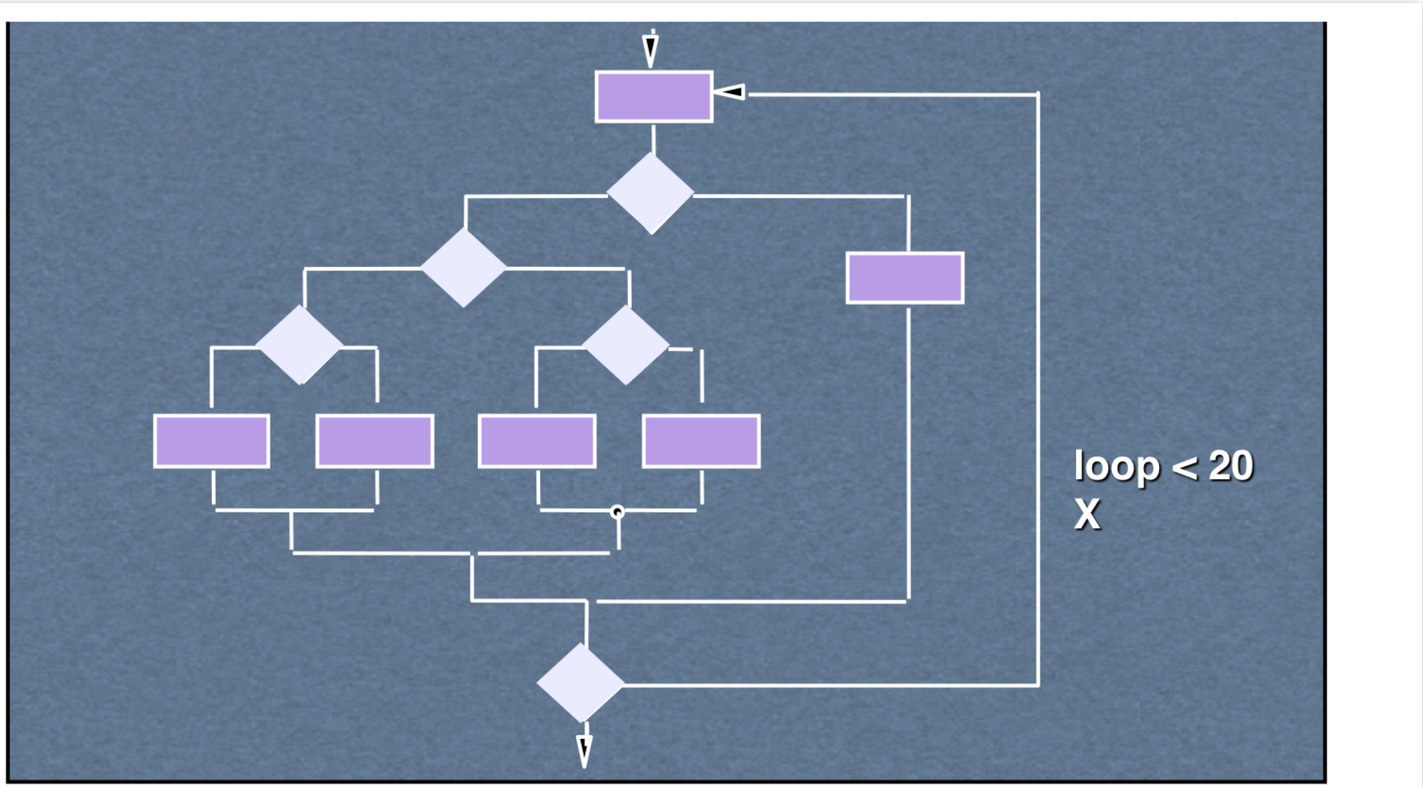
- Agile processes rarely use formal inspection or peer review processes.
- Rather, they rely on team members cooperating to check each other's code, and informal guidelines, such as 'check before check-in', which suggest that programmers should check their own code.
- Extreme programming practitioners argue that pair programming is an effective substitute for inspection as this is, in effect, a continual inspection process.
- Two people look at every line of code and check it before it is accepted.

# Execution-based Testing

- "The process of ***inferring*** certain behavioral properties of a product based, in part, on results of executing the product in a ***known environment*** with ***selected inputs***"

Testing is the process of exercising a program with the specific intent of finding errors prior to delivery to the end user

# Exhaustive Testing




**There are $10^{14}$ possible paths! If we execute one test per millisecond, it would take 3,170 years to test this program!!**

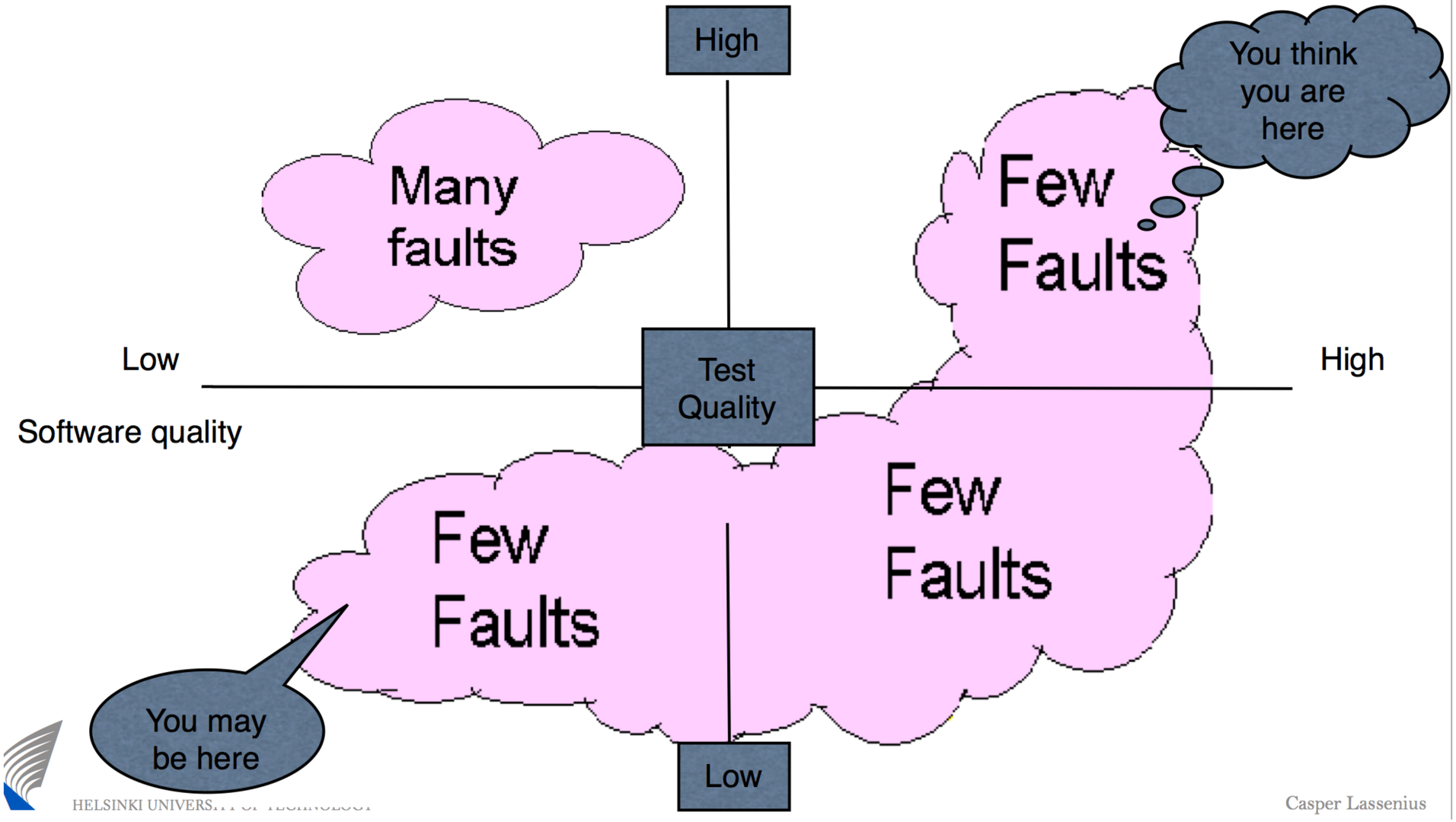
# Who Should Test the Software?

- Developer
  - Understands the system
  - Driven by “delivery”
  - Tests “gently”
- Independent tester
  - Must learn about the system
  - Driven by quality
  - Will attempt to break the system
- Customer
  - Acceptance testing

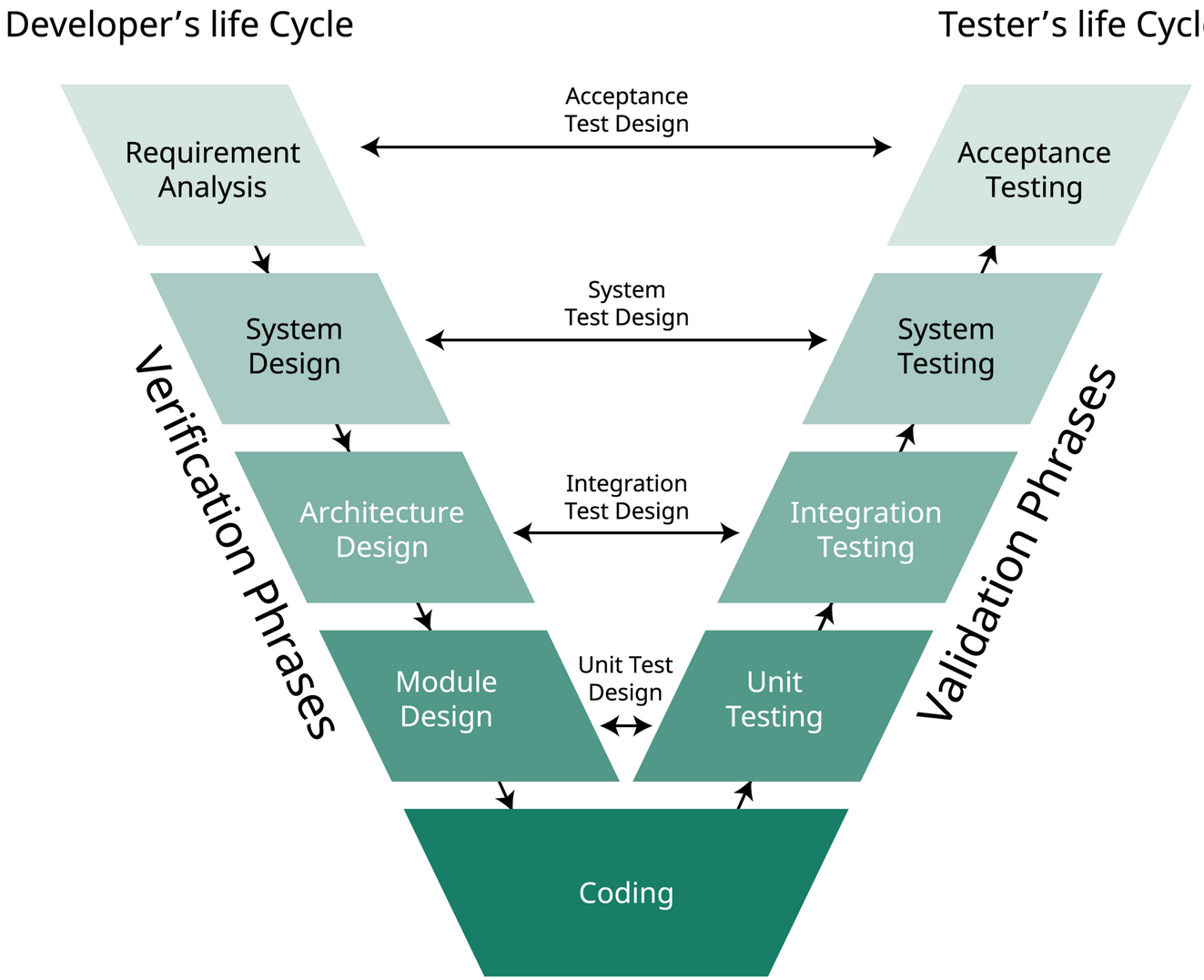
# Testing — Some Observations

- A professional programmer produces ~6 faults/1000 lines of code
- New program with 200 000 LOC has ~1200 faults
- Program that have been in use for long have ~1 fault/1000 LOC
- Error removal costs ca 12h/fault

# Assessing Software Quality



# Testing Strategy: The V-model



# 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.

# Prioritizing Tests

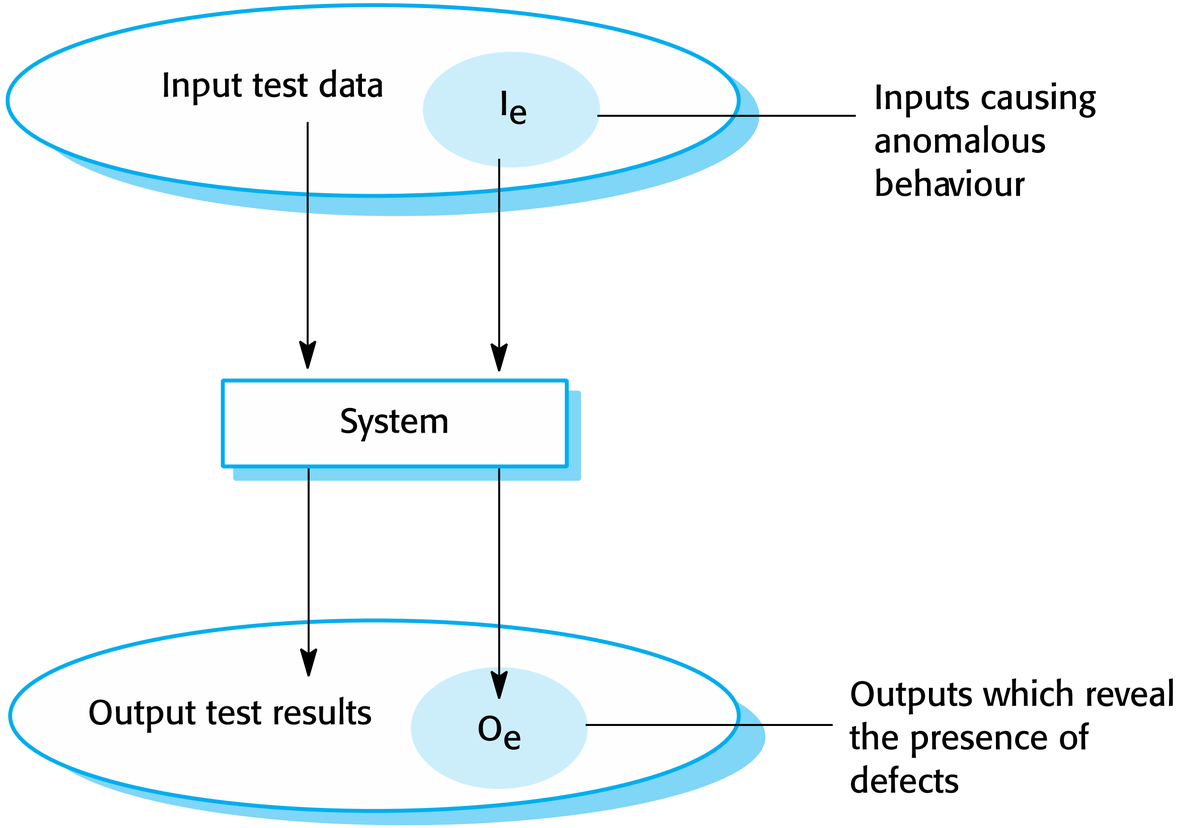
- Time is always limited
- Use risk to focus testing effort
  - what to test first
  - what to test most
  - how thoroughly to test each feature
  - what not to test (at least for now)
- Most important tests first

- Possible ranking criteria
- test where a failure would be most severe
- test where failures would be most visible
- test where failures are most likely
- ask the customers to prioritise the requirements
- what is most critical to the customer's business
- areas changed most often
- areas with most problems in the past
- most complex or technically critical areas

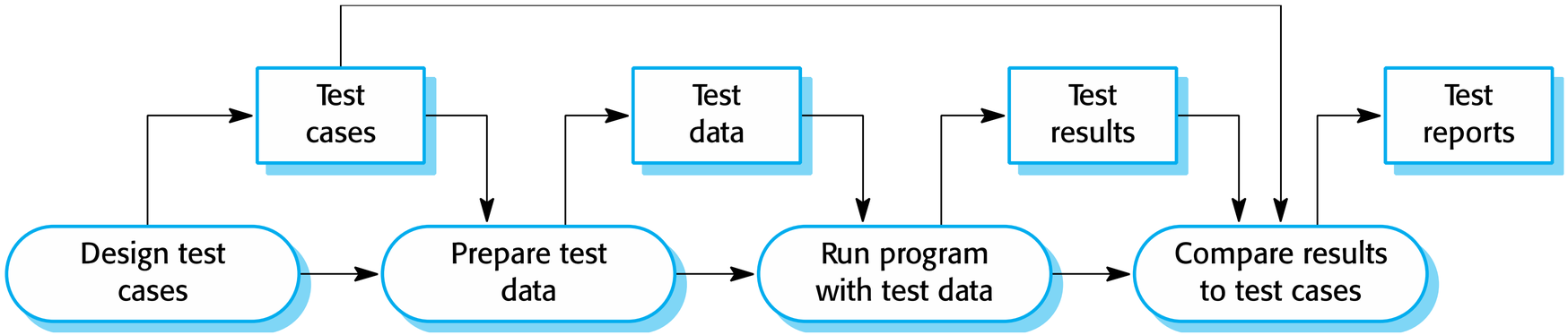
# Realities in Software Testing

- Testing can show the presense of errors but cannot show their absense
- All bugs cannot be found
- Testing does not create quality software or remove defects
- Not all bugs found will be fixed
- Testing focuses on critiquing the product, not the developer(s)

# An input-output model of program testing



# A model of the software testing process



# Stages of testing

- Development testing, where the system is tested during development to discover bugs and defects.
- Release testing, where a separate testing team test a complete version of the system before it is released to users.
- User testing, 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**, where individual program units or object classes are tested. Unit testing should focus on testing the functionality of objects or methods.
    - **Component testing**, where several individual units are integrated to create composite components. Component testing should focus on testing component interfaces.
    - **System testing**, where some or all of the components in a system are integrated and the system is tested as a whole. System testing should focus on testing component interactions.

# Unit testing

- Unit testing is the process of testing individual components in isolation.
- It is a defect testing process.
- 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.

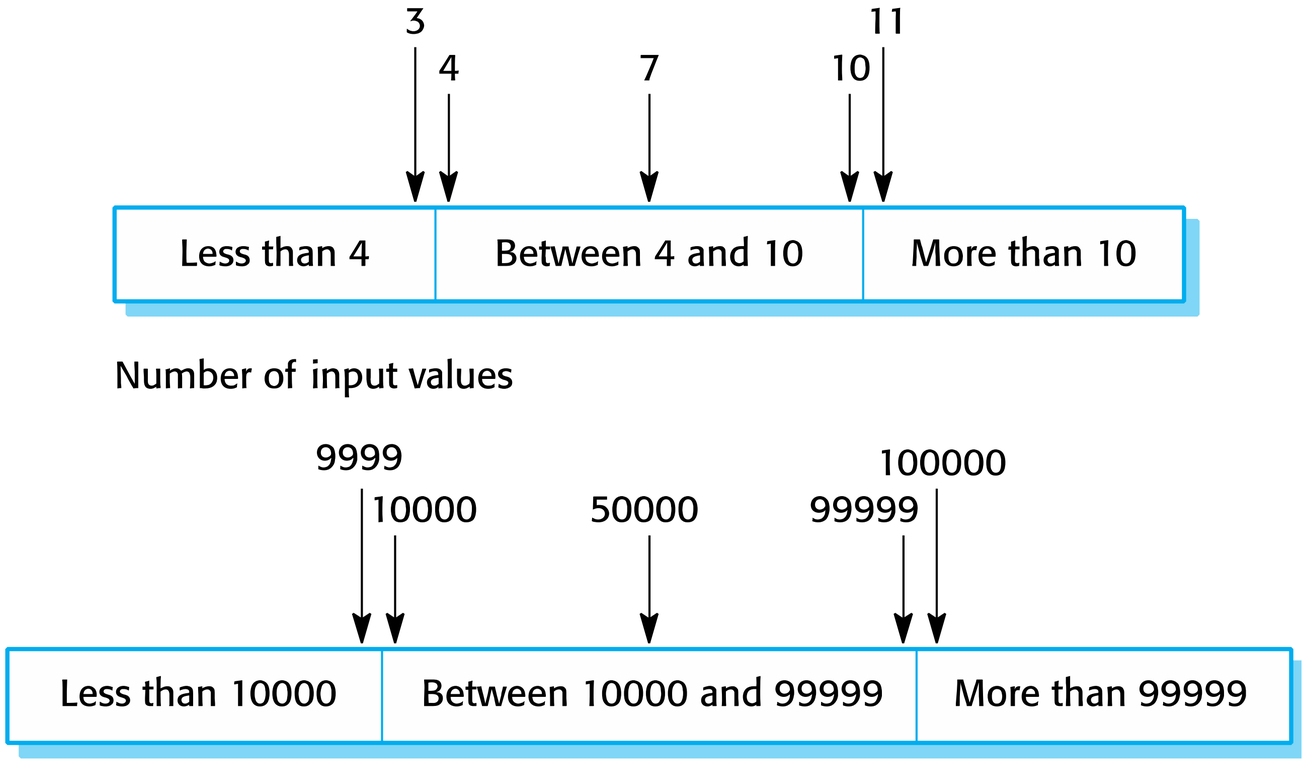
# Automated testing

- 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.

# 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.

# Equivalence partitions

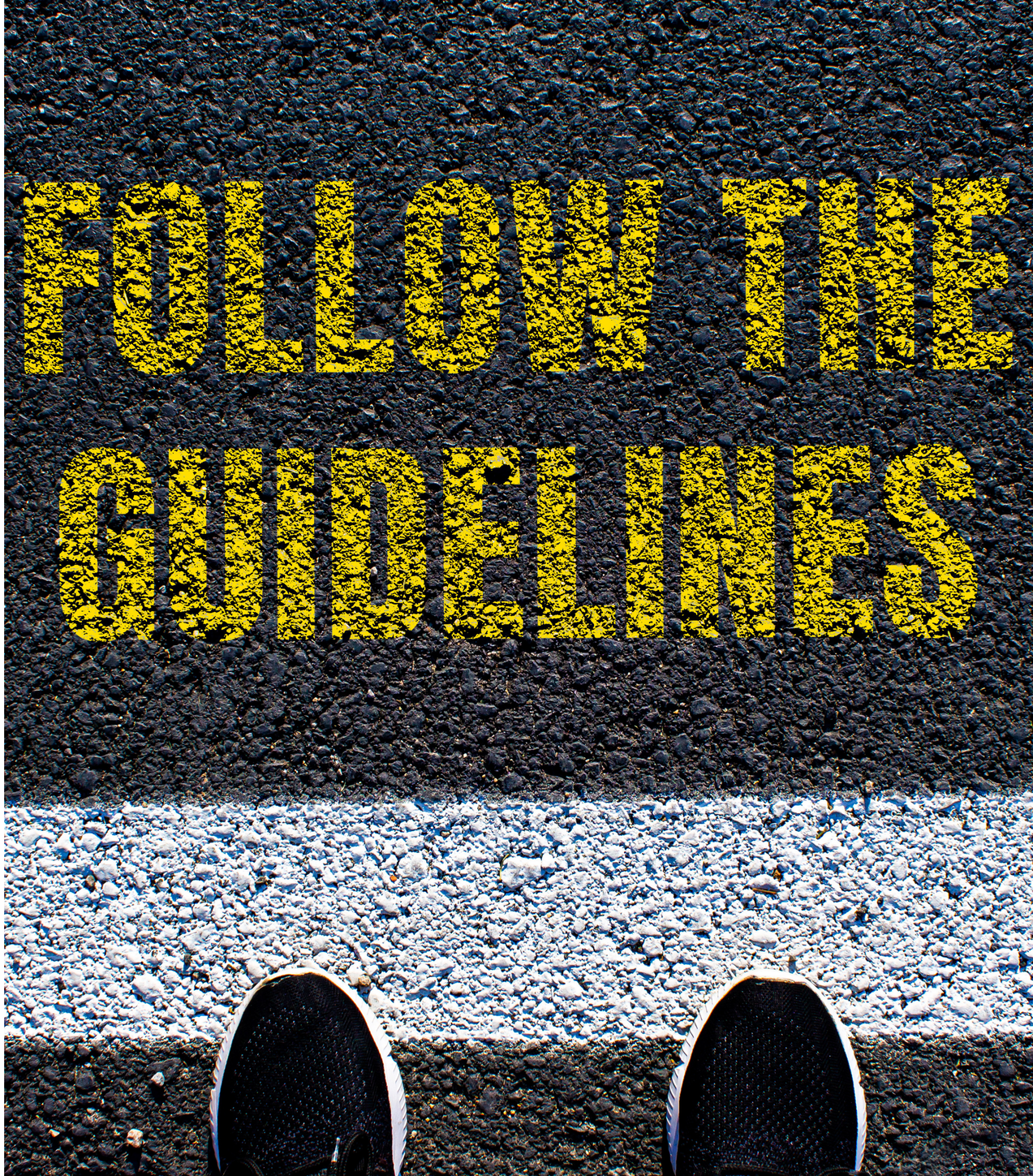


# Testing guidelines (sequences)

- 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

- 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.



# System testing

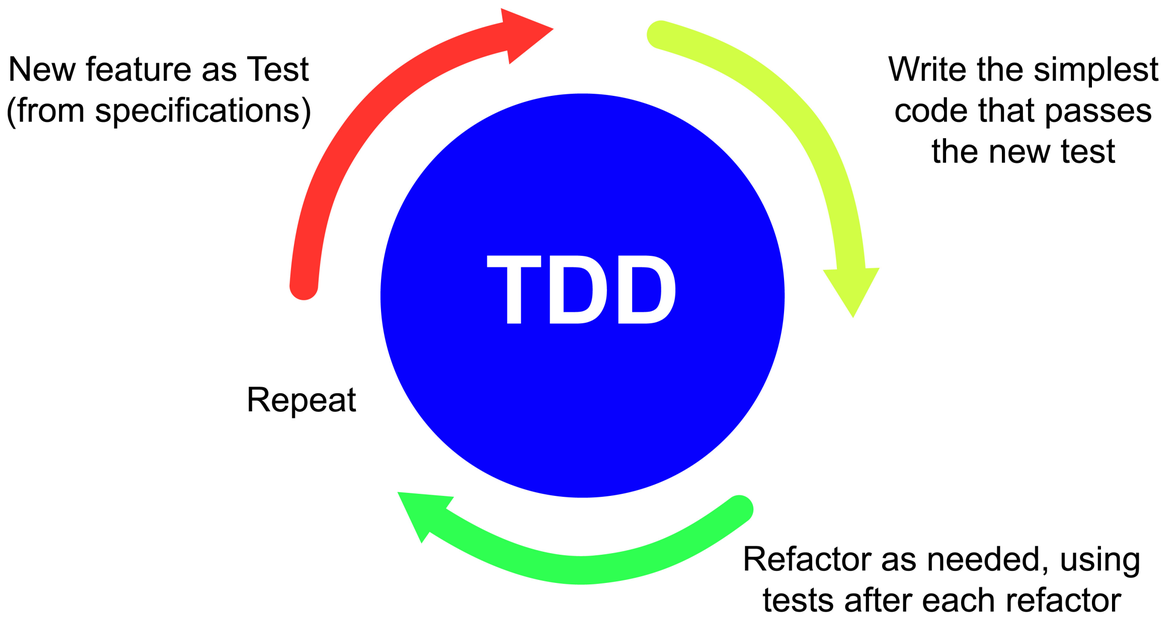
- System testing during development involves integrating components to create a version of the system and then testing the integrated system.
- The focus in system testing is testing the interactions between components.
- System testing checks that components are compatible, interact correctly and transfer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

# Use-case testing

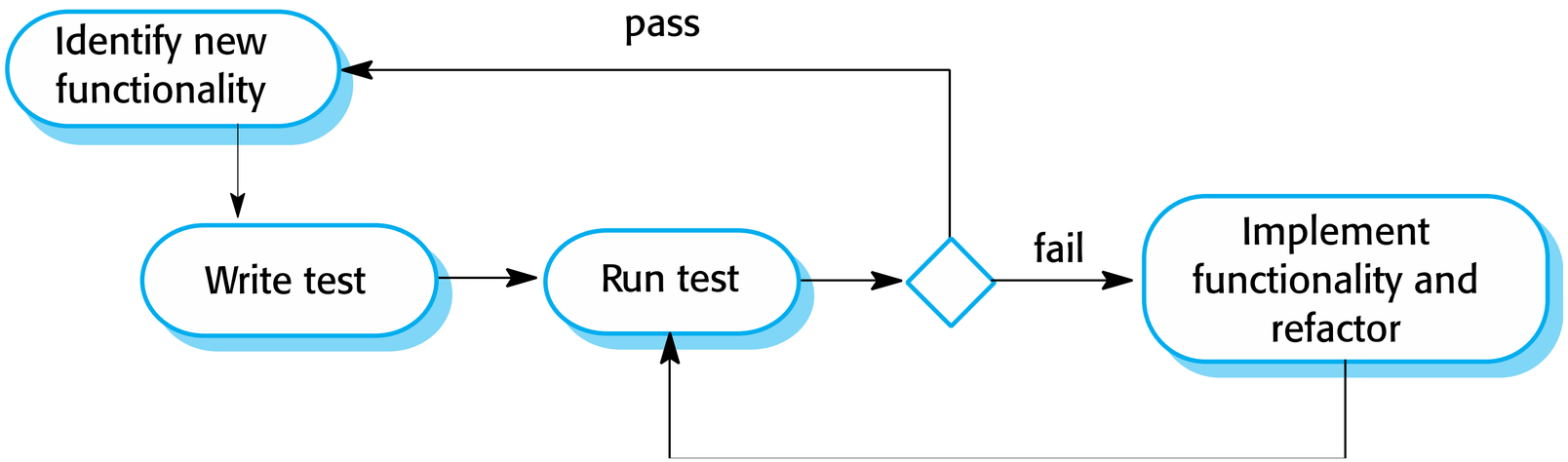
- The use-cases developed to identify system interactions can be used as a basis for system testing.
- Each use case usually involves several system components so testing the use case forces these interactions to occur.
- The sequence diagrams associated with the use case documents the components and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

# Test-driven development

- Test-driven development (TDD) is an approach to program development in which you inter-leave testing and code development.
- Tests are written before code and ‘passing’ the tests is the critical driver of development.
- You develop code incrementally, along with a test for that increment. You don’t move on to the next increment until the code that you have developed passes its test.
- TDD was introduced as part of agile methods such as Extreme Programming. However, it can also be used in plan-driven development processes.

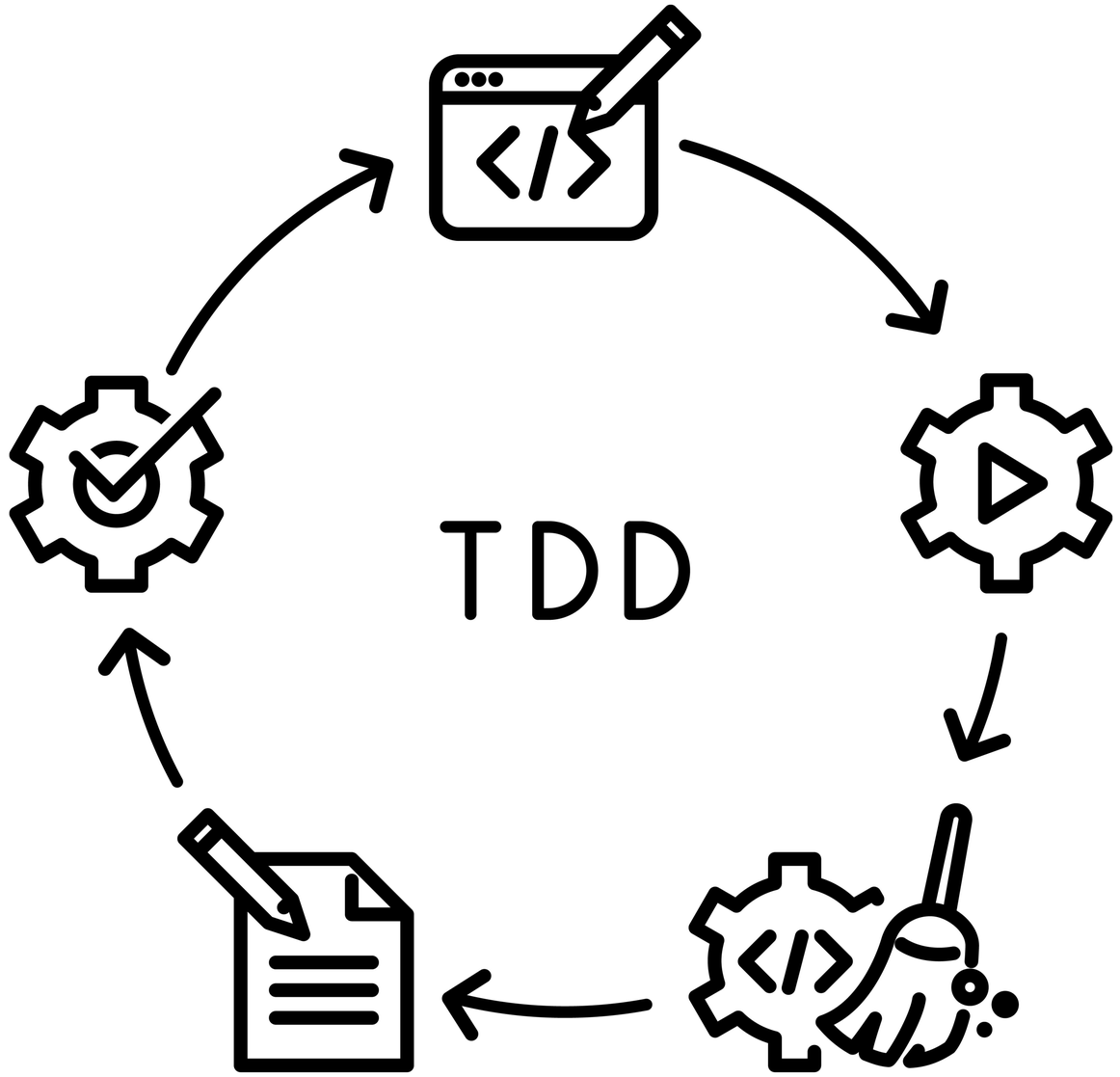


# Test-driven development



# TDD process activities

- Start by identifying the increment of functionality that is required. This should normally be small and implementable in a few lines of code.
- Write a test for this functionality and implement this as an automated test.
- Run the test, along with all other tests that have been implemented. Initially, you have not implemented the functionality so the new test will fail.
- Implement the functionality and re-run the test.
- Once all tests run successfully, you move on to implementing the next chunk of functionality.



# Benefits of test-driven development

- Code coverage
  - Every code segment that you write has at least one associated test so all code written has at least one test.
- Regression testing
  - A 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. The newly written code needs to be checked and modified.
- System documentation
  - The tests themselves are a form of documentation that describe what the code should be doing.

# Regression testing

- Regression testing is testing the system to check that changes have not ‘broken’ previously working code.
- In a manual testing process, regression testing is expensive but, with automated testing, it is simple and straightforward. All tests are rerun every time a change is made to the program.
- Tests must run ‘successfully’ before the change is committed.

# 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.
  - Release testing, therefore, has to show that the system delivers its specified 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).

# Performance testing

- Part of release testing may involve testing the emergent properties of a system, such as performance and reliability.
- Tests should reflect the profile of use of the system.
- 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.

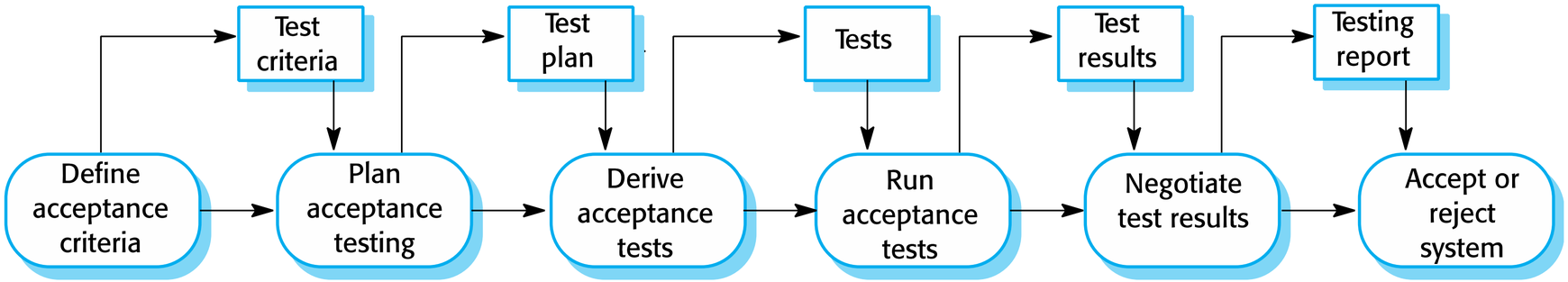
# User testing

- 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 reason for this is that influences from the user's working environment have a major effect on the reliability, performance, usability and robustness of a system. These cannot be replicated in a testing environment.

# 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
  - Customers test a system to decide whether or not it is ready to be accepted from the system developers and deployed in the customer environment. Primarily for custom systems.

# The acceptance testing process



# Agile methods and acceptance testing

- In XP, the user/customer is part of the development team and is responsible for making decisions on the acceptability of the system.
- Tests are defined by the user/customer and are integrated with other tests in that they are run automatically when changes are made.
- There is no separate acceptance testing process.
- Main problem here is whether or not the embedded user is ‘typical’ and can represent the interests of all system stakeholders.

# Re-testing

- Run a test, it fails, fault reported
- New version of software with fault “fixed”
- Re-run the same test (i.e. re-test)
  - must be exactly repeatable
  - same environment, versions (except for the intentionally changed software)
  - same inputs and preconditions
- If test now passes, fault has been fixed correctly—or has it?

# Smoke Testing

- A common approach when using “daily builds”
- Smoke testing steps
  - Software components are integrated into a build
    - A build includes all data files, libraries, reusable modules, and engineered components that are required to implement one or more features
  - A series of tests is designed to expose errors
    - The intent is to uncover “show stopper” errors that have the highest likelihood of throwing the project behind schedule
  - The build is integrated with other builds and the entire product is smoke tested daily