CS5030

Software Testing

Learning objectives

- On completing this lecture and associated reading, you should
  - Understand the purpose of testing
  - Be aware of how testing fits in the context of verification and validation
  - Know a set of basic testing methods
  - Understand the limitations of testing

Testing

- “Program testing can be used to show the presence of bugs, but never to show their absence!”

  - Edsger Dijkstra

- “Testing is about producing failures”

  - Bertrand Meyer

Software testing

- Testing is intended to
  - show that a program does what it is intended to do. and
  - discover program defects before it is put into use
- During testing, software is executed, typically using artificial data
- The results of the test run are checked for errors, anomalies or information about non-functional attributes
- Testing is part of a more general verification and validation process

Verification and validation

- 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 requires
- They can be used to establish whether the system is fit for purpose

Goals of testing - 1

- To demonstrate to the developer and the customer that the software meets its requirements
  - For custom software
    - there should be at least one test for every requirement in the requirements document
  - For generic software products
    - there should be tests for all the system features, and combinations of these features, that will be part of the product release
  - The system is expected to perform correctly using a given set of test cases that reflect the system’s expected use

Goals of testing - 2

- To discover situations in which the behaviour of the software is incorrect, undesirable or does not conform to its specification
- Defect testing is concerned with finding
  - undesirable system behaviour such as system crashes,
  - unwanted interactions with other systems,
  - incorrect computations and
  - data corruption
- Test cases are designed to expose defects
  - Can be deliberately obscure and need not reflect how the system is normally used

Other V &V methods

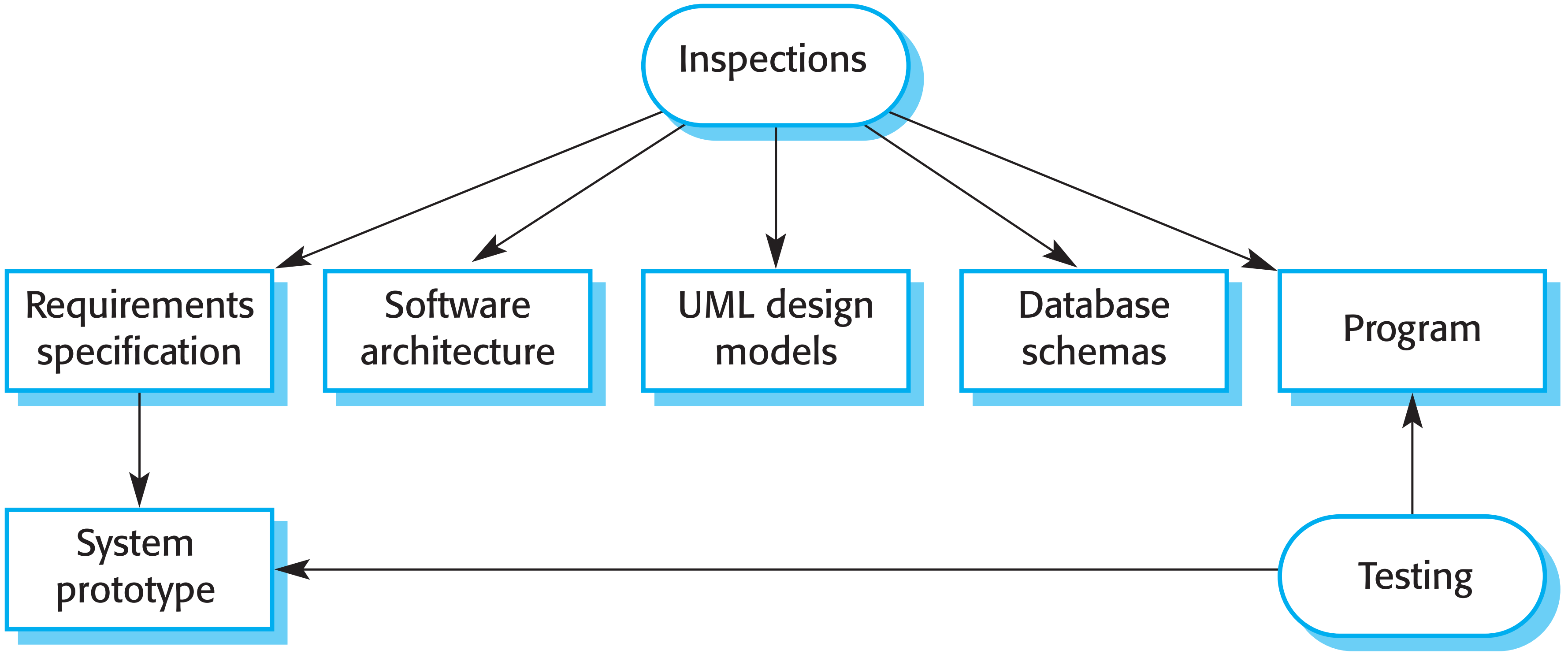
- Walk-through
- Code review
- Inspection
- Formal methods

Inspections vs testing

- Software inspections
  - Concerned with analysis of the static system representation to discover problems (static)
  - May be supplemented by tool-based document and code analysis

- Software testing
  - Concerned with exercising and observing product behaviour (dynamic)
  - The system is executed with test data and its operational behaviour is observed

Inspections vs testing



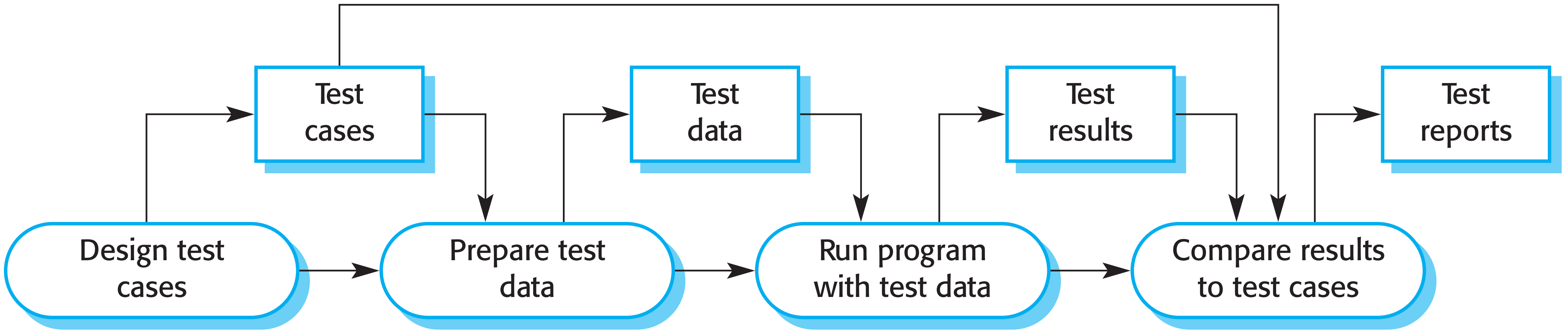
[Sommerville, 2016]

Inspections vs testing

- Advantages of inspections over testing
  - During testing, some errors can mask other errors
  - Specialised test harnesses are required to test incomplete software
  - Testing is not suitable for determining certain quality attributes

- Disadvantages of inspections over testing
  - Inspections are not suitable for finding errors due to unexpected interactions between parts of a system and timing / performance issues
  - Small companies or teams may not be able to find an independent inspection team

Software testing process



[Sommerville, 2016]

Types of testing - when

- Stage of the lifecycle
  - Development
    - The system is tested during development to discover bugs and defects
  - Release
    - A separate testing team tests a complete version of the system before it is released to users
  - Production
    - The system is tested after it is deployed

Types of testing - who

- Developers
- Test engineers, quality assurance team
- Domain experts
- Users
    - Users or potential users of a system test the system in their own environment

Types of testing - what

- Functional
  - Unit, component, interface, integration, system, regression, acceptance

- Non-functional
  - Performance (including stress, load, scalability), security, usability, accessibility, …

Types of testing - how

- Knowledge of internal structure
  - Opaque (PKA black box)
  - Transparent (PKA white box)
  - Partially transparent (PKA grey box)

- Automation
  - Manual vs automated

Development testing

- All testing activities carried out by the team developing the system
  - Unit testing
  - Component testing
  - System testing
  - Acceptance testing (for agile development)
  - … etc

Unit testing

- Unit testing is the process of testing individual program units in isolation
  - focus on testing the functionality

- Units may be:
  - Individual functions or methods within an object
  - Classes with several attributes and methods
  - Composite components with defined interfaces used to access their functionality

- It is a defect testing process

Unit test cases

- Test cases should show that, when used as expected, the component being tested does what it is supposed to do
- If there are defects in the component, these should be revealed by test cases
- 2 types of unit test case
  - Reflecting normal operation of a program showing that the component works as expected
  - Based on scenarios where common problems arise, using abnormal inputs to check that these are properly processed and do not crash the component

Testing strategies

- Partition testing
  - identify groups of inputs that have common characteristics and should be processed in the same way
  - choose tests from within each of these groups

- Path-based testing
  - ensure each path through the code under test is executed at least once

- Guideline-based testing
  - use testing guidelines to choose test cases
  - guidelines reflect previous experience of the kinds of errors that programmers often make when developing components as well as available domain knowledge

Component testing

- Software components are often composite
  - made up of other interacting components
  - functionality of components is accessed through well-defined component interface

- Testing composite components
  - focus on showing that the component interface behaves according to its specification
  - assuming that unit tests on the individual components /objects within the component have been completed

Interface testing

- Aims 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

System testing

- System testing involves
  - integrating components to create a version of the system, and
  - testing the integrated system as a whole
- Focus is on 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
- Testing the emergent behaviour of a system

System and component testing

- Integration of components prior to system testing
    - reusable components that have been separately developed,
    - off-the-shelf systems / components, and
    - newly developed components

- Components developed by different team members or sub-teams may be integrated at this stage
    - Collective rather than an individual process
    - In some companies, system testing may involve a separate testing team with no involvement from designers and programmers

System testing - drivers

- Requirements / use cases
  - Opaque - need to know only requirements; at least one test per requirement
- Structure
  - Transparent - determines whether elements of systems work correctly
- Statistics
  - Assesses trustworthiness via testing with randomly sampled input data
- Risk
  - Based on identified risks, aims to ensure that the system is not vulnerable
- Level of coverage achieved

System 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 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

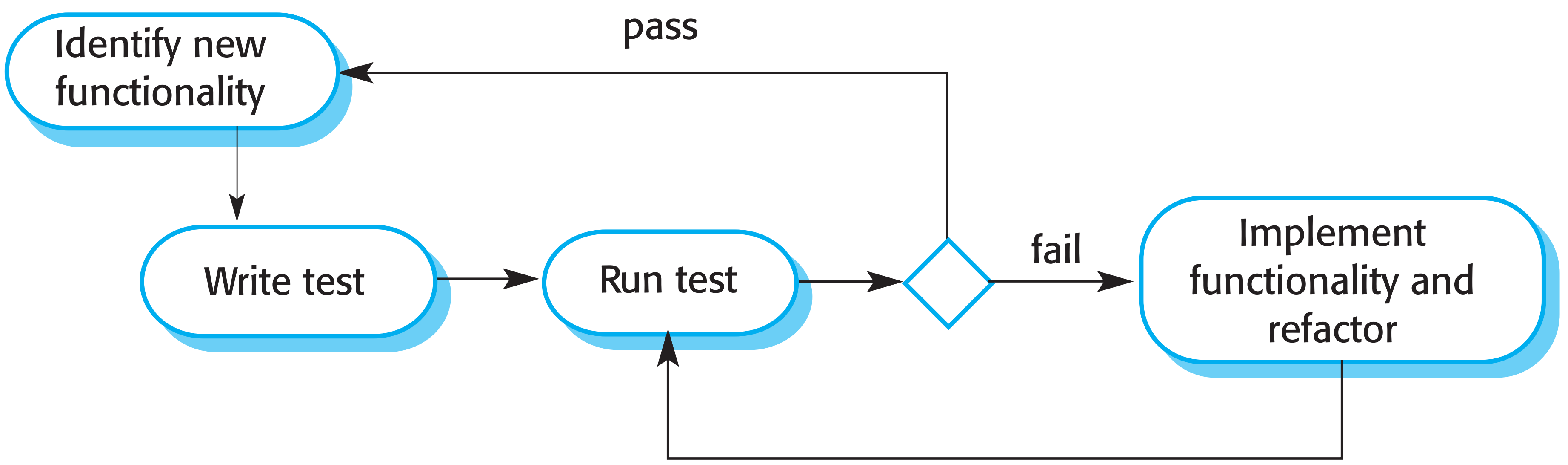
Acceptance testing

- In agile development, acceptance tests are not only run at the end of development just before delivery and sign-off of a contract
- Instead, they are
  - specified in a formal document shared by the stakeholders and the development team
  - written in a formal language that is nevertheless understandable for all parties involved
- Acceptance tests provide a way to define “done”

Test-driven development – TDD (1)

- An approach to program development in which testing and code development are interleaved
- Tests are written before code and passing the tests is the critical driver of development
- Code is developed incrementally, along with a test for that increment
  - Cannot move to the next increment until the code that has been 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 (2)


Identify new functionality
pass
Write test
Run test
fail
Implement functionality and refactor

[Sommerville, 2016]

Benefits of TDD

- Code coverage
  - Every code segment has at least one associated 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
- System documentation
  - The tests themselves are a form of documentation that describes what the code should be doing

Regression testing

- Testing the system to check that changes have not broken previously working code
- Cost of regression testing
  - Expensive with a manual testing process
  - Simple and straightforward with automated testing
    - All tests are rerun every time a change is made to the program
- Tests must run successfully before the change is committed

Release testing

- 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
  - 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 an opaque testing process where tests are only derived from the system specification

Release testing vs 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
  - For eg, 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
  - Abnormal / anomalous conditions

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
  - Users’ working environment can 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

Acceptance testing and agile methods

- In agile methods, 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
- One potential problem here is whether or not the embedded user is typical and can represent the interests of all system stakeholders

Costs of testing

- The cost of fixing errors in a system rises over time through successive phases of the development process
- Continuous testing allows errors to be caught early, reducing the costs of fixing them
- Test early, test often
- A system should have a suite of tests at different granularities
  - Avoid test duplication
- Automate as much as possible

Testing – case study

- Description and example

- The practical test pyramid
  - Martin Fowler

Limitations of testing

- As Dijkstra said, testing can only show the presence of errors
- Formal verification methods are still too costly to be applied to many software systems, so systematic testing is still most common

Test quality

- Tests are as important as application software
- Tests need to be maintained as application is maintained

Key points

- Software testing is the most widely used verification method
- Testing can find errors but not prove their absence
- We can distinguish between testing practices on the basis of when, by whom, what, how and to what extent testing is done
- Test-driven development is a key component of most agile software development approaches
- Acceptance tests provide a way to define ‘done’
- Most forms of testing can be automated and so can be done early and often
- Formally specified tests replace extensive system specification documents in agile approaches