V&V Processes
Plan and Track vs Test Driven Development

Software Requirements and Project Management
CITS3220
Overview

- **Plan and Track Testing** *(traditional SE)*
  - Unit and Integration tests
  - System tests: functional
  - System test: performance
  - Acceptance and Installation Tests

- **Test Driven Development** *(agile SE)*
  - Testing is part of design and development; not something that follows as an extra
  - Test early; Test often; Test automatically
Plan and Track
The image shows a flowchart illustrating the testing process of a software system. The flowchart includes the following stages:

1. **Design specifications**
2. **System functional requirements**
3. **Other software requirements**
4. **Customer requirements specification**
5. **User environment**

Each stage leads to the next, eventually leading to the system's installation test. The flowchart includes steps such as:

- **Unit test**
- **Integration test**
- **Function test**
- **Performance test**
- **Acceptance test**
- **Installation test**

The flowchart indicates the progression from unit testing to system testing, culminating in the system being put into use.

**Key Points**

- Integrated modules
- Functioning system
- Verified, validated software
- Accepted system

**Note:** The diagram also highlights the importance of component code and tested components in the overall testing process.
Integration Test Techniques

- Once satisfied that units are correct, we combine them into the full system and test the combination.
- In order to manage fault-detection for the full system we need an \textit{incremental integration} strategy.
- Some possible integration test strategies:
  - bottom-up, top-down, big-bang, sandwich
  - see Bruegge & Dutoit (2\textsuperscript{nd} ed) Sect 11.4.4
Having tested that the programs satisfy the design, we must now test whether the system does what the customer wants.

System function tests check that the integrated system performs the functions specified in the requirements.

System performance tests check all non-functional properties.

System tests are usually black box tests.
System Function Tests

- Function tests can be generated from a system specification using black box unit test methods

- For example, *domain testing* and *conformance test* methods can also be used on a full system
System tests for Performance and Reliability

- Performance Tests compare integrated components with non-functional system requirements
  - security, accuracy, speed and reliability
  - usually test HW as well as SW

- NOT just performance=speed
Types of Performance Test

- stress
- volume
- configuration
- compatibility
- regression
- security
- timing
- environmental (on industry site)
- quality
- recovery
- maintenance
- documentation
- human factors (usability)
Performance Tests (1)

- **Stress Testing** checks if the system can respond to many simultaneous requests.

- **Volume Testing** attempts to find faults associated with large amounts of data, such as static limits imposed by the data structures, high-complexity algorithms or high disk fragmentation.
Performance Tests (2)

- **Security Testing** attempts to find security faults.
- There are few systematic methods for doing this.
- Usually this test is achieved by “tiger teams” who attempt to break into the system, using their experience and knowledge of typical security flaws.
Performance Tests (3)

- **Timing tests** attempt to find behaviours that violate timing constraints

- **Recovery tests** evaluate the ability of the system to recover from errors, such as the unavailability of resources, a hardware failure or a network failure
Acceptance Tests

- is the customer’s test against the requirements
- **pilot test** installs the system on an experimental basis
- **alpha test** is an in-house pilot test by the developers
- **beta test** is the customer’s pilot test
- **parallel test** runs the new system alongside the old
Installation Test

- run in the system’s actual operational environment

- tests focus on the **completeness** of the installed system and verification of any functional or non-functional characteristics that may be affected by site conditions e.g. severe weather, background radiation etc.

- installation tests include re-runs of some tests passed in the lab. to check results are still OK in the field
Regression Testing

- As SW is tested and used, faults are discovered that need correction
- A new release of SW is an improved system which replaces the previous one
- Regression testing identifies new faults that may have been introduced as current ones are being corrected
- When planning tests you should also plan for regression tests
Test Documents

- The *Test Plan* focuses on managerial aspects of testing: scope, approach, resources, and schedule of test activities.
- Each test case is documented by a *Test Case Specification*.
- Each test execution is documented by a *Test Incident Report* that records the actual results of the tests and differences from expected output.
- All the failures which were discovered during the tests and that need to be investigated are documented in a *Test Report Summary*. 
Test Plan Outline

1. Introduction objectives and extent of tests
2. Relationship to other documents eg. requirements, design
3. System Overview components which are tested
4. Features to be tested / not to be tested and the reasons
5. Pass/Fail criteria generic criteria for the tests in plan
6. Approach eg. reasons for selected integration strategy
7. Suspension and resumption criteria and repeats
8. Testing materials (HW/SW requirements)
9. Test Cases the core of the plan, lists the test cases used
10. Testing schedule responsibilities, staffing and training, risks, contingencies and the test schedule
Test Case Specification

Outline

1. Test case specification identifier
2. Test items
3. Input specifications
4. Output specifications
5. Environmental needs
6. Special Procedural requirements
7. Inter-case dependencies
Test Incident & Summary Reports

- **Incident Report** lists actual test results and the failures that were experienced. Which features were demonstrated and whether the features has been met. If failure, sufficient information to allow the failure to be reproduced.

- **Failures** from all Test Incident Reports are collected and listed in the Test Summary Report and then further analyzed and prioritized by the developers.
Test Driven Development

Notes from

Kevlin Henney

kevlin@curbralan.com
Traditional View of Testing

- Validation and verification of code internal quality and requirement satisfaction
  - Carried out after the fact, although in principle based on a test plan drawn up beforehand
- Separated from design and coding
  - Including separation of the people involved
  - Encourages a pipeline-style of development process
Traditional Reality of Testing

- Testing is seen generally as a dull activity
  - An obstacle between the joy of code and the satisfaction of delivery

- Testing is perceived as destructive
  - A negative rather than a positive activity and role

- Testing is often performed...
  - late, superficially, manually and with interruptions from bugs
Testing vs Debugging

- Debugging cannot be considered a sustainable development activity
  - It can rarely be estimated with any sense of reality or confidence
  - Reliance on debugging as a form of quality assurance is amusing... if you're not paying

- Long feedback cycles in development amplify the need for debugging
  - And discourage code quality
A Better Practice

- Testing is a part of development
  - Not something that follows it as an extra
- Testing needs to be carried out at all levels at which a system is conceived
  - Programmers responsible for code-level tests
  - System testing is a different role and activity

Test Early. Test Often. Test Automatically.

*Andrew Hunt and David Thomas*
Test-Driven Development

- TDD has emerged from the many practices that form Extreme Programming's core
  - Focused on code-centric practices in the micro process rather than driving the macro process
- TDD can be used in other macro-process models
  - TDD is not XP, and vice versa
  - TDD is not just unit testing
Core Practices

- **Intent**
  - Present TDD in terms of its defining practices

- **Content**
  - Test-bounded design increments
  - Programmer testing responsibility
  - Automated tests
  - Example-based test cases
  - Active test writing
  - Refactoring
Test-Bounded Design Increments

- Design is often a speculative activity whose outcome can be more complex than necessary
  - Tests define requirements at the code level, as opposed to the system level
  - Tests offer a practical stopping condition on the scope and complexity of a design step
- Clear development steps encourage a certain sufficiency and minimalism in design
Programmer Testing Responsibility

- If testing is carried out only at system or subsystem level, defects accumulate
  - Which leads to a long, debug-based feedback cycle between system testers and programmers

- However, having another person unit test a programmer's code is also wasteful
  - Long, communication-based feedback cycles

- Therefore, programmers are responsible for testing what they write
Automated Tests

- The primary focus of TDD is programmer tests on code units
  - Although the meaning is often extended to include the customer-focused tests that define a macro-process development increment
- A prerequisite for making a continuous testing approach viable is automation
  - Testing happens as part of a build, not as a manual task carried out separately and on the whole program
Example-Based Test Cases

- The structure of individual test cases is to demonstrate functionality by example
  - Therefore, the tests are functional as opposed to performance tests, or other operational tests
- Tests focus on and use the interface of a unit and not its internal structure
  - Specific examples with specific values are used rather than generating the test case based on code's internal structure or testing exhaustively
Active Test Writing

- Test-driven development is based on writing tests with the code
  - A test case should be written just before, alongside or just after the functionality it tests

- Integrated into the lifecycle
  - Frequent testing gives timely feedback on code usability, coupling and sufficiency, so testing has a qualitative as well as a quantitative side
  - Testing acts as a driver rather than a passenger
Refactoring

- Refactoring is personal hygiene for software
  - Often predicated on the existence of unit tests for unit-level regression testing
  - Unit tests also support optimisation activities
- Remove unnecessary or confusing aspects of the code base — remove to improve
  - Do so iteratively and decrementally
  - The less code you have, the less you have to test, fix, debug, optimise, ...