Chapter 8 Notes

Goal of Testing is to show that a program does what it is intended to do and to discover program defects before it is put into use.

Testing Processes have 2 Goals

* Demonstrate (to the developer ane customer) that the SW meets its requirements.
  + For Customer SW, this means that there should be at least on etest for every requirement
  + For Generic SW, this means that there should be tests for all system features and a combination of these services
  + This goal leads to Validation Testing
* To discover situations in which the behavior of the SW is incorrect, undesirable, or does not conform to its specification. These are consequences of SW defects.
  + This goal leads to Defect Testing

Validation Testing

* Where you expect the system to perform correctly using a given set of test cases that reflect the system’s expected use.

Defect Testing

* Test cases are designed to expose defects
* Test cases in defect testing can be deliberately obscure and need not reflect how the system is normally used

Testing cannot demonstrate that SW is free of defects or that is will behave as specified in every circumstance

* “Testing can only show the presence of errors, not their absence”

Testing is part of a broader process of SW Verification and Validation.

* Validation
  + Are we building the right product?
  + Is more general than Verification
  + Ensures that the SW meets the customer’s expectations.
* Verification
  + Are we building the product right?
  + Checks that the SW meets its stated functional and non-functional requirements.

V&V processes are concerned with checking that SW being developed meets its specification and delivers the functionality expected by the customer.

* Goal of V&V
  + To establish confidence that the SW system is “fit for purpose”. The level of required confidence depends on:
    - SW Purpose
      * More critical SW -> more important is that it be reliable
    - User expectations
      * At first users have low expectations because of their experience with buggy SW. Later they develop higher expectations and thus expect the SW to become more reliable
    - Marketing Environment
      * If the SW is cheap, then users will tolerate bugs
      * If it is a competitive environment then early release of the system is appropriate (if even with bugs) so that it can be the 1st system in the market

Static V&V Techniques

* Don’t need to execute the SW to verify it
* V&V may also involve inspections and reviews to analyze and check the system requirements, design models, source code, and proposed system tests.
  + 3 Advantages of Inspection over Testing
    - During testing, errors can hide other errors. When an error leads to unexpected outputs, you can never be sure if later output anomalies are due to a new error or are side effects of the original error.
    - Incomplete version of a system can be inspected without additional costs.
    - Inspection can also consider broader quality attributes of a program, such as compliance with standards, portability, and maintainability. Can look for inefficiencies, inappropriate algorithms, and poor programming style.
  + Disadvantages
    - Not good for discovering defects that arise because of unexpected interactions between different parts of the program, timing problems, or problems with system performance.
    - In small teams, it can be difficult and expensive to put together a separate inspection team

SW Testing process

* Design Test Cases
  + Output: Test Cases
* Prepare Test Data
  + Input: Test Cases
  + Output: Test Data
* Run program with Test Data
  + Input: Test Data
  + Output: Test Results
* Compare Results to Test Cases
  + Input: Test Results and Test Cases
  + Output: test Reports

3 Stages of Testing

* Development Testing
  + System is tested during development to discover bugs and defects
* Release Testing
  + Separate testing team tests a complete version of the system before it is released to users
* User Testing
  + Users and potential users of a system test the system in their own environment.

Testing involves both manual and automated testing.

Development Testing (Section 8.1)

* Includes all testing activities that are carried out by the team developing the system
  + Tester of the SW is usually the programmer who developed that SW, although this is not always the case.
  + For more critical systems, this can be more formal with a separate testing group within the development team.
* Testing may be carried out at 3 levels of granularity
  + Unit Testing
    - Individual program units or object classes are tested. Focus on testing the functionality of objects or methods
  + Component Testing
    - Several individual unites are integrated to create composite components. Focuses on testing component interfaces
  + System Testing
    - Some or all of the components are integrated and the system is test as a whole. Focuses on testing component interfaces
* Development Testing is primarily a defect testing process (aim is to discover bugs in the SW).
* Unit Testing (Section 8.1.1)
  + Process of testing program components, such as methods or object classes. Individual functions or methods are the simplest type of component.
  + Test should call these routines with different input parameters.
  + When testing object classes, tests should provide coverage of all of the features of the object:
    - Test all operations with the object
    - Set an check the value of all attributes associated with the object
    - Put the object into all possible states. You should simulate all events that cause a state change.
  + Whenever possible, try to automate Unit Testing
    - Frameworks like JUnit write and run your unit tests.
    - An automated test has 3 parts
      * A setup part, where you initialize the system with the test case, namely the inputs and expected outputs.
      * A call part, where you call the object or method to be tested
      * An assertion part, where you compare the result of the call, with the expected result. If the assertion evaluates to true, the test is successful and fails otherwise.
  + You the unit you are testing depends on another object (like a database) you can make and create a mock object.
  + Types of test cases
    - Partition Testing
      * Identify groups of inputs that have common characteristics and should be processed in the same way. You should chose test from within each of these groups
    - 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.
  + Partition Testing
    - Input data and output results of a program often fall into a number of different classes with common characteristics. Examples of classes are positive numbers, negative numbers, and so on. Programs normally behave in a comparable way for all members of a class -> Because of this sometimes these classes are called Equivalence Partitions or Domains.
      * Once you identify a set of partitions, you choose test cases from each of these partitions.
      * A good rule of thumb for test case selection is to choose test cases on the boundaries of the partitions, plus cases close to the midpoint of the partition. (Reason for this is that programmers tend to consider typical values of input when developing a system).
  + Guideline Testing
    - Encapsulate knowledge of what kinds of test cases are effective for discovering errors. For example, when testing programs with array, sequences, lists ,etc try the following:
      * Test eh SW with sequences that have only a single value
      * Use different sequences of different sizes in different tests
      * Derive tests so that the first, middle, and last elements of the sequence are accessed.
    - Other guidelines include
      * 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 input numerous times
      * Force invalid outputs to be generated
      * Force computation results to be too large or too small.
* Component Testing (Section 8.1.3)
  + Focus is on testing that the component interfaces behave according to specifications
  + Types of interfaces
    - Parameter interfaces
      * Where data or sometimes function references are passed from one component to another. Methods in an object have a parameter interface.
    - Shared memory interfaces
      * Interfaces in which a block of memory is shared between components. Data is palced in the memory by one subsystem and retrieved from there by other sub-systems.
      * Used in embedded sytems
    - Procedural interfaces
      * One component encapsulates a set of procedures that can be called by other components
    - Message passing interfaces
      * One component requests a service from another component by passing a message to it
  + Types of interface errors
    - Interface misuse
      * A calling component calls some other component and makes an error in the use of its interface. Parameters may be of the wrong type or there are too many parameters.
    - Interface misunderstanding
      * Calling component misunderstands the specification of the interface of the call component and makes assumptions about its behavior.
    - Timing errors
      * Producer of data and the consumer of data may operate at different speeds. Unless particular care is taken in the interface design, the consumer can access out-of-data information because the producer has not updated.
  + Guidelines for Interface Testing
    - Examine the code to be tested and explicitly list each call to an external component. Design a set of tests in which the values of the parameters to the external components are at the extreme ends of their ranges. These extreme values are most likely to reveal interface inconsistencies.
    - Where pointers are passed across an interface, always test the interface with null pointer parameters
    - Where a component is called through a procedural interface, design tests that deliberately cause the component to fail. Differing failure assumptions are one of the most common specification misunderstandings
    - Use stress testing in message passing systems. Generate many more messages than are likely to occur in practice
    - Where several components interact through shared memory, design tests that vary the order in which these components are activiated.
* System Testing (Section 8.1.4)
  + Involves integrating components to create a version of the system and then testing the integrated system.
  + System testing checks that components are compatible, interact correctly and transfer the right data at the right time across their interfaces.
  + Differences with Component Testing
    - In System Testing, components that have been separately developed and off the shelf system may be integrated with newly developed components. The complete system is then tested
    - Component developed by different team members or groups may be integrated at this stage. System testing is collective rather than an individual process
  + Some elements of system functionality only become obvious when you put the components together.
  + System Testing should focus on testing the interactions between the component and objects that make up a system
  + Use Case Testing is effective here because of the need to test interactions
    - Use Cases forces interactions to occur

Test-Driven Development (Section 8.2)

* Is an approach to program development in which you interleave testing and code development
* You develop code incrementally, along with a test for that increment. You don’t move on to the next increment until the code you have developed passes its test.
* It was part of Agile Method but it can also be used in Plan-Driven development
* TDD Process
  + Start by identifying the increment of functionality that is required. Usually small and implementable in few lines of code
  + Write a test for this functionality and implement this as an automated test.
  + Runt the test, along with other tests that have been implemented. Initially, you have not implemented the functionality so the new test will fail. This is deliberate as it shows that the test adds something to the test set.
  + You then implement the functionality and re-run the test. This may include refactoring existing code to improve it and add new code to what’s already there.
  + Once all tests run successfully, move on to implementing the next chunk of functionality
* Using frameworks like JUnit you can every test every time that you add functionality.
* Advantages
  + Helps programmers clarify their ideas of what a code segment is actually supposed to do.
  + To write a test, you need to understand what is intended, as this understanding makes it easier to write the required code.
  + Code coverage
    - Every code segment that you write should have at least one associated test. Therefore you can be confident that all of the code in the system has actually been executed.
  + Regression Testing
    - A test suite is developed incrementally as a program is developed. You can always run regression tests to check that changes to the program have not introduced new bugs.
  + Simplified debugging
    - When a test fails, it should be obvious where the problem lies. You do not need to use debugging tools to locate the problem.
  + System documentation
    - Tests themselves act as a form of documentation that describe what the code should be doing.
* Negatives
  + If you are reusing large code components or legacy systems then you need to write tests for these systems as a whole.
  + TDD may be ineffective in multi-threaded systems.
  + Still need to provide a separate Validation Testing process

Release Testing (Section 8.3)

* Is the process of testing a particular release of a system that is intended for use outside of the development team
* 2 Distinction between Release Testing and System Testing
  + 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)
* Goal of Release Testing
  + 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.
* Requirements-Based Testing (Section 8.3.1)
  + Requirements should be testable…A requirement should be written so that a test can be written for it
  + Is a systematic approach to test case design where you consider each requirement and derive a set of tests for it.
  + Is Validation Testing…not Defect Testing – You are trying to demonstrate that the system has properly implemented its requirements
* Scenario Testing (Section 8.3.2)
  + Is an approach to release testing where you devise typical scenarios of use and use these to develop test cases for the system.
  + A scenario is a story that describes one way in which the system might be used. They should be realistic and real system use should be able to relate to them.
    - A release tester would go through this story acting as the user and note how the system behaves in response to different inputs.
* Performance Testing (Section 8.3.3)
  + Once a system has been completely integrated, it is possible to test for emergent properties, such as performance and reliability.
  + Performance tests have to be designed to ensure that they system can process its intended load.
  + Is concerned with demonstrating that the system meets its requirements and discovering problems and defects in the system.
  + To test whether performance requirements are being achieved, you may have to construct an operational file
    - Operational File
      * Is a set of tests that reflect the actual mix of work that will be handled by the system. Ex. 90% of transaction in a system are of type A, 5% of type B, and the remainder types C, D, and E.
  + An effective way to discover defects is to design tests around the limits of the system.
    - In Performance Testing, this means stressing the system by making demands that are outside the design limits of the software.
    - This type of testing has 2 functions
      * It test failure behavior of the system (i.e. does the system corrupt data upon failure)
      * It stresses the system and may cause defects to come to light that would not normally be discovered.
  + This testing is important in distributed system which experience severe degradation when they are heavily loaded.
    - The network gets swamped with coordination data that the different processes must exchange. The processes become slower and slower as they wait for required data from other processes.

User Testing (Section 8.4)

* Is where users of customers provide input and advice on system testing
  + This may involve formally testing a system that has been commissioned from an external supplier, or could be an informal process where users experiment with a new software product to see if they like lit and that it does what they need.
* Is essential because influences from the user’s working environment have a major effect on the reliability, performance, usability, and robustness of the system. It is impossible for the developer to replicate the system’s working environment.
* 3 Types of User Testing
  + Alpha Testing
    - Users of the SW work with development team to test the SW at the developers site
    - Often used when developing SW products that are sold as shrink-wrapped systems.
  + Beta Testing
    - A release of the SW is made available to users to allow them to experiment and to raise problems that they discover with the system developers.
    - Takes place when an early, sometimes unfinished, release of a software system is made available to customers and users for evaluation.
    - Can be publicly available or selectively available
    - Used for SW that are used in many different environments
    - Helps with marketing
  + Acceptance Testing
    - Customer test a system to decide whether or not it is ready to be accepted from the system developers and deployed in the customer environment
    - Takes place after Release Testing.
    - Involves a customer formally testing a system to decide whether or not it should be accepted from the developer.
    - 6 stages in the Acceptance Testing Process
      * Define acceptance criteria
        + Should take place early in the process before the contract for the system is signed
      * Plan acceptance testing
        + Deciding on the resources, time, and budget for acceptance testing and establishing a testing schedule.
        + Should define risks to the testing process, such as system crashes and inadequate performance, and discuss how these risks can be mitigated
      * Derive acceptance tests
        + Once acceptance criteria have been established, tests have to be designed to check whether or not a system is acceptable.
        + Acceptance tests should aim to test both the functional and non-functional characteristics of the system.
      * Run acceptance tests
        + The agreed acceptance tests are executed on the system.
      * Negotiate test results
        + It is very unlikely that all of the define acceptance tests will pass and that there will be no problems with the system. IF this is the case, the testing is finished and the system can be handed over. If there are problems, the developer and the customer have to negotiate to decide if the system is good enough to be put into use. They must also agree on the developer’s response to identified problems
      * Reject/Accept System
        + Involves a meeting between the developers and the customer to decide on whether or not the system should be accepted. If rejected, then further development is required.
    - There is no separate Acceptance Testing process in XP programming since the user is PART of the development team and has been making tests the entire time.

Key Points

* Testing can only show the presence of errors in a program.
* Development testing is the responsibility of the SW development team. A separate team should be responsible for testing a system before it is released to customers. In the User Testing process, customer and users provide test data and check that tests are successful.
* Development testing includes Unit Testing, in which you test individual objects and methods; Component Testing which you test related groups of objects; and System Testing, in which you test partial or complete systems.
* When testing SW, you should try to “break” the SW by using experience and guidelines to choose types of test cases that have been effective in discovering defects in other systems
* Wherever possible, you should write automated tests.
* Test-first development is an approach to development where tests are written before the code to be tested. Small code changes are made and the code is refactored until all tests execute successfully.
* Scenario testing is useful because it replicates the practical use of the system. It involves inventing a typical usage scenario and using this to derive test cases
* Acceptance Testing is a user testing process where the aim is to decide if the SW is good enough to be deployed and used in it operational environment.