Testing and Evolution

CSCI 201

Principles of Software Development

Jeffrey Miller, Ph.D.
jeffrey.miller@usc.edu

USC Viterbi
School of Engineering

Outline

- Testing
- Evolution

Software Testing

- The testing process has two goals
  - To demonstrate to the developer and the customer that the software meets its requirements
  - To discover situations in which the behavior of the software is incorrect, undesirable, or does not conform to its specification
- Testing cannot demonstrate that the software is free of defects or that it will behave as specified in every circumstance
  - Dijkstra stated that “testing can only show the presence of errors, not their absence”

Validation and Verification

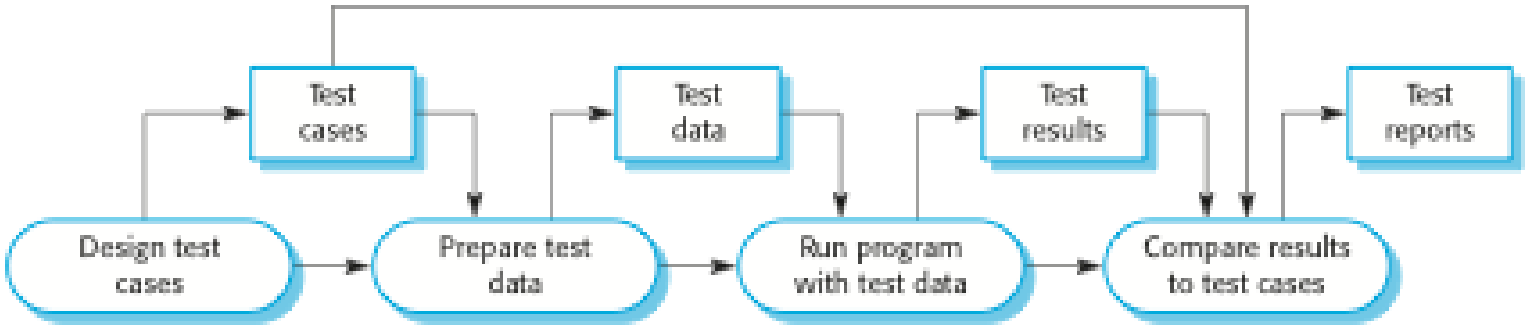
- Barry Boehm stated the difference as
  - Validation: Are we building the right product?
  - Verification: Are we building the product right?
- V&V processes are concerned with checking that software being developed meets its specification and delivers the functionality expected by the people paying for the software

Software Inspections

- V&V involves software inspections and reviews, which have three advantages over solely testing
  - With testing, errors can mask other errors, but inspection can discover multiple errors
  - Incomplete versions of a system can be inspected
  - Inspections can also consider broader quality attributes of a program, such as compliance with standards, portability, and maintainability

Testing Stages

- A commercial software system will go through three stages of testing
  - Development Testing – tests during development that 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 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
- There are three levels of granularity for development testing
  - Unit Testing – where individual program units or object classes are tested
  - Component Testing – where several individual units are integrated to create composite components
  - System Testing – where some or all of the components in a system are integrated and the system is tested as a whole

Development Testing – Unit Testing

- Unit testing is the process of testing program components, such as methods or object classes
- Unit tests should show:
  - When used as expected, the component you are testing does what it is supposed to
  - Defects in the component should be revealed
- When designing test cases, you should:
  - 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

Development Testing – Unit Testing

- With all types of testing, there are black box and white box testing
- Black box testing uses the specifications of a system to identify the test cases
- White box testing looks at the code to identify additional test cases

Unit Testing Example

- What inputs should we use to test the following code?

```
// this method will read int values from the user and return
// an int array that contains those values
int[] readArray() {
  int[] numArray = new int[10];
  Scanner scan = new Scanner(System.in);
  int i=0;
  int num = scan.nextInt();
  while (num != -1) {
    numArray[i++] = num;
    num = scan.nextInt();
  }
  return numArray;
}
```

Test Cases

Valid inputs: 1, 2, 3, 4, -1

Too many inputs: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, -1

No inputs: -1

Large int input: 123456789123456789123456, -1

Small int input: -123456789123456789123456, -1

Non-int inputs: 1.2, 3.4, 4.5, -1

String inputs: hello, goodbye, -1

Control character inputs: \0, \n, \t, -1

Development Testing – Component Testing

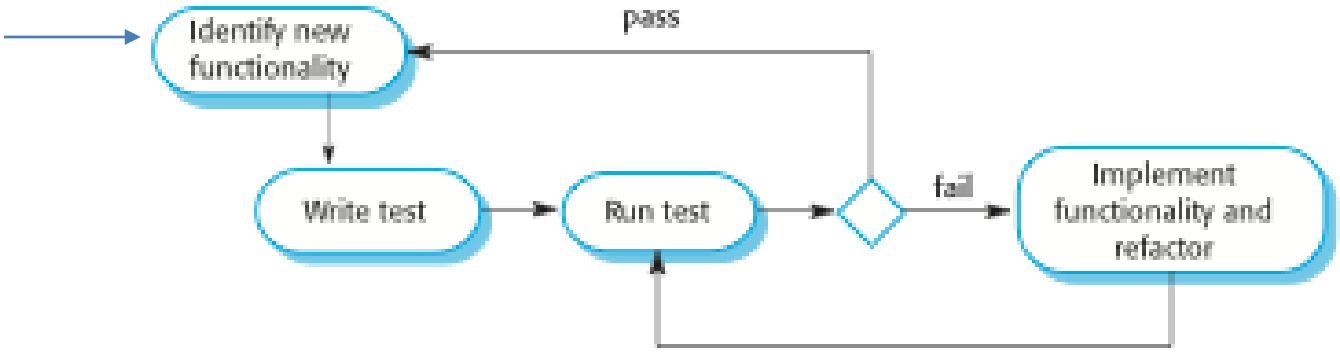
- Software components are composite components that are made up of several interacting objects
- Since unit testing should have already found errors in the individual components, the component testing should focus on the interfaces between the components
- Here are some of the interfaces
  - Parameter Interfaces – interfaces in which data or function references are passed from one component to another
  - Shared Memory Interfaces – interfaces in which a block of memory is shared between components
  - Procedural Interfaces – interfaces in which one component encapsulates a set of procedures that can be called by other components
  - Message Passing Interfaces – interfaces in which one component request a service from another component by passing a message to it

Development Testing – System Testing

- System testing during development involves integrating components to create a version of the system and then testing the integrated system
- There is some overlap with component testing, but two differences exist
  - During system testing, reusable components that may have been separately developed or COTS products may be integrated with the other components
  - Components developed by different team members or groups will be integrated at this stage

Test-Driven Development

- Test-Driven Development (TDD) is an approach to program development in which you interleave testing and code development
  - TDD was introduced as part of agile methodologies (i.e. Scrum, XP)
  - TDD greatly reduces the cost of regression testing



Release Testing

- Release testing is the process of testing a particular release of a system that is intended for use outside the development team
- There are two distinctions between release testing and system testing
  1. A separate team that has not been involved in the system development should be responsible for release testing
  2. Release testing checks that the system meets its requirements and is good enough for external use

Release Testing Types

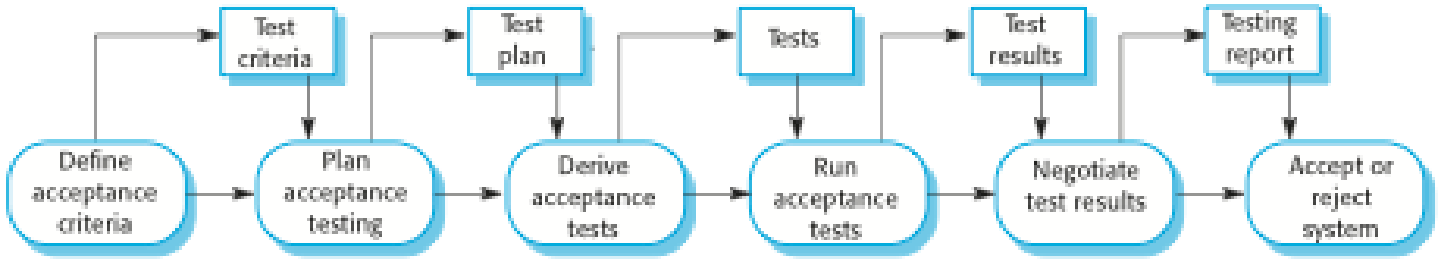
- Requirements-based testing provides one or more test cases for each requirement
  - Based on good requirements engineering, this should be possible
- Scenario testing devises typical scenarios of use of the system, then develops test cases based on them
- Performance testing ensures that the system can process its intended load
  - These are often called load tests or stress tests as well

User Testing

- User testing is a stage in the testing process where users or customers provide input and advice on system testing
- There are three types of user testing
  - Alpha Testing – users 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 they discover with the developers
  - Acceptance Testing – customers test a system to decide whether it is ready to be accepted from the developers and deployed in the customer environment

User Testing – Acceptance Testing

- There are six stages in the acceptance testing process
  - Define acceptance criteria
  - Plan acceptance testing
  - Derive acceptance tests
  - Run acceptance tests
  - Negotiate test results
  - Reject or accept system

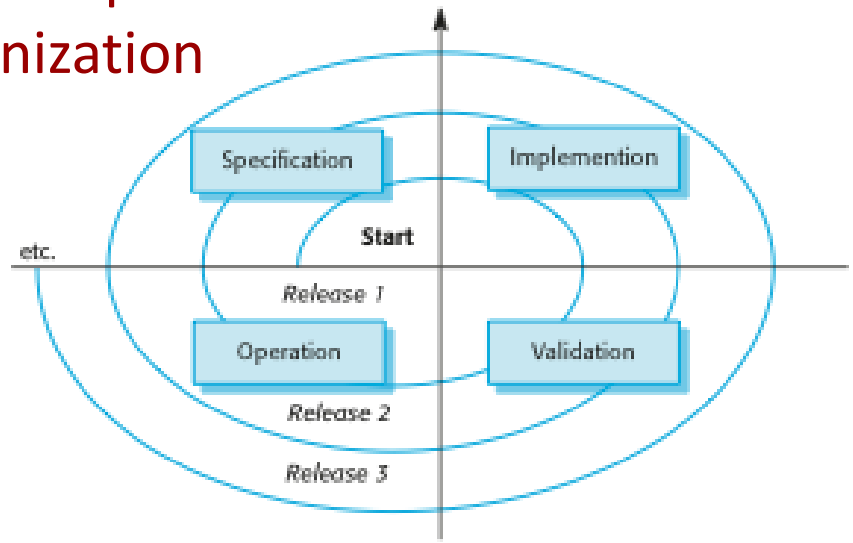


Outline

- Testing
- Evolution

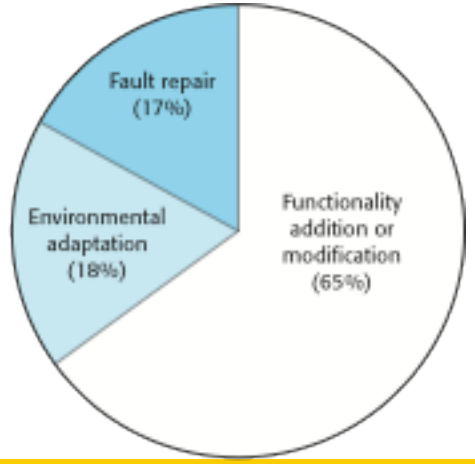
Software Evolution

- Even after a system is delivered, it will inevitably have to change to remain useful
- Software is expensive, so organizations need to use it for a number of years to get a return on their investment
  - This often results in multiple releases of software so the system evolves with the organization



Software Maintenance

- Software maintenance is the general process of changing a system after it has been delivered
- There are three different types of software maintenance
  - Fault Repairs – coding errors are relatively cheap to repair, design errors are more expensive, and requirements errors are the most expensive
  - Environmental Adaptation – hardware, OS, or other support software changes, causing the system to be modified to adapt
  - Functionality Addition – when the system requirements change in response to organizational or business change
- Distribution of the maintenance costs



Functionality Addition

- It is usually more expensive to add functionality after a system is in operation than it is to implement the same functionality during development because
  - Team Stability – the development team may be broken up and reassigned after a system has been delivered
  - Poor Development Practice – a maintenance contract may be given to a different company than the original development company
  - Staff Skills – maintenance staff are often inexperienced and unfamiliar with the application domain
  - Program Age and Structure – as programs age, they become harder to understand and change since changes to programs tend to degrade the program structure