Disciplined Software Testing Practices

Dr. Magdy Hanna

Chairman

International Institute for Software Testing

Sponsored by:

International Institute for Software Testing

Practice #1
Requirements are Crucial for Good Testing

- You can't test what you do not know
- Users will always change their minds
- Requirement precision and completeness is a must
- The use of prototyping or RAD is NOT an excuse for not documenting requirements
- Keep in mind: GUI describes only high level functional requirements; no details; no business rules; no quality requirements
- Rule of thumb: Users do not know what they want until they do not get it

Practice #2
Test for Both Functional and Quality Requirements

- Although functional requirements seem to be most important to the user, most software disasters result from poor quality software.
- Quality requirements are the least understood requirements by both customers and developers.
- Although never talked about, quality requirements seem to be an “assumed” customer expectation.
- You can’t achieve quality unless you define it.
- You can’t test for quality unless you quantify it.

The Sixteen Fitness-for-Use Factors (1)

1. Configurability

the ability to configure the software for user's convenience. Examples are changing the system-user interface to use certain graphical symbols or changing the default use of directories.

2. Correctness

the degree to which the software conforms to the user's functional requirements.

3. Efficiency

the amount of resources required by the system to perform its intended functionality. This includes processing time, memory, disks, or communication lines.

4. Expandability

the ability to change the software to add more functionality or to improve its performance. This deals with the perfective maintenance of the software

The Sixteen Fitness-for-Use Factors (2)

5. Flexibility

the ability to change the software to function in a different environment. This includes working under different database structure or to execute in a context different from that considered in the original implementation. This deals with the adoptive maintenance of the software.

6. Integrity

the ability of the software to protect itself and its data from overt and covert access.

*7. I**nteroperability***

the ability of the software to exchange data with other software

8. Maintainability

the ability to change the software to fix errors. This deals with the corrective maintenance of the software.

The Sixteen Fitness-for-Use Factors (3)

9. Manageability

the ability to properly manage the administrative aspects of the software. This includes resource allocation, configuration management, etc. It also deals with the availability of tools to support manageability.

10. Portability

the ability of the software to function on different platforms.

11. Reliability

the rate of failures in the software that make it unable to deliver its intended functionality.

12. Reusability

the ability to reuse portions of the software in other applications.

The Sixteen Fitness-for-Use Factors (4)

13. Safety

the ability of the software to perform its functionality without causing any unsafe conditions.

14. Survivability

the ability of the software to continue execution, even with degraded functionality, after a software or hardware failure.

15. Usability

the ease by which the software can be learned and used.

16. Verifiability

the ease by which it can be verified that the software is working correctly.

Practice #3
Adopt Model-Based Requirements

- Natural language text is not a reliable way of specifying requirements.
  - Ambiguous
  - Vague
  - Incomplete
  - Difficult to communicate and understand
  - Not easily testable

Models for Specifying Requirements

- Use any of the following models to augment natural language text:
  - Data Models
  - Process Models
  - Object Models
  - State Models
  - User Interface Models
  - Decision Tables
  - Decision Trees
  - Use Cases

These will be covered in other courses!

Practice #4
Formally Define Test Cases

- For each requirement, identify all possible scenarios.
- Consider input data, internal state, and external events
- Use techniques such as
  - Equivalence Classes
  - Cause and Effect graphs
  - Decision Tables
  - Decision Trees
  - Use Cases
  - State Models
- For each scenario, determine all test cases using techniques such as Boundary Value Conditions

Ad Hoc (Informal) Testing

- Unplanned testing is bad if it is your only strategy!
  - Monkey testing
  - Beating the keys
  - Just try a little of this mentality
- Can you accept accidental quality?
  - What functional areas did you cover?
  - What is the risk of shipping this build?
  - What is the general quality of the SW?

Remember that one output of the test effort is gaining knowledge [measure the quality] of the SW.

Practice #5:
Perform Positive and Negative Testing

- **Positive Testing** is any activity aimed at evaluating an attribute or a capability of a system and determine that it meets its requirements. Expectation is that data inputted is ***valid*** and will be processed through the normal paths.
- **Negative Testing** is the process of executing a program or system with the intent of finding errors. Expectation is that data inputted is ***invalid*** and will be processed through the error handling paths.

Practice #6
Trace Requirements to Test Components (Test Cases or Test Scenarios)

	Req 1	Req 2	Req 3	Req 4
Test Case 1	Passed/ Failed	Passed/ Failed		X
Test Case 2		Passed/ Failed	Passed/ Failed	
Test Case 3			Passed/ Failed	X
Test Case 4	Passed/ Failed			X
Test Case n				

Practice #7

Trace Test Cases to Database Tables

	TC 1	TC 2	TC 3	TC 4	TC n
Table 1	U		U		
Table 2	R	R	C	C	
Table 3	U	R	R		
Table 4		D	D		
Table 5	D	D	D		
Table 6	R/U	D			
Table n					

C → Create R → Read U → Update D → Delete

Practice #8
Perform More Thorough Regression Testing

- Base your regression test suite on:
  - Impact analysis
  - Risk analysis
- Use of capture/playback tools can help

Remember: Current users of the SW can be economically harmed if the new version does not give them the functionality they already depend on!

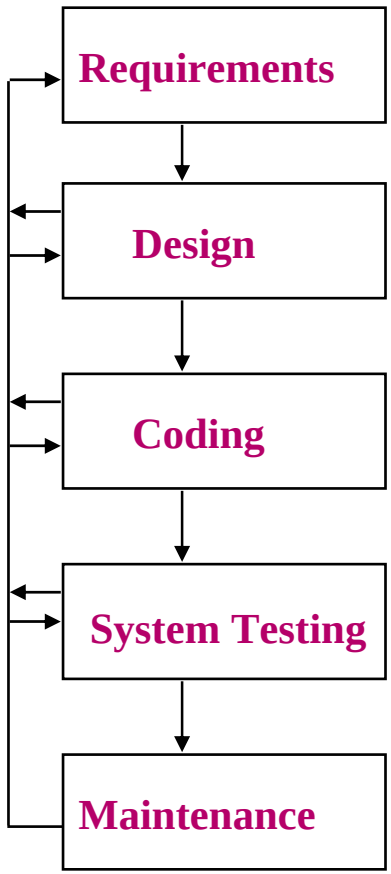
Perform Impact Analysis

- When a requirement is changed, what components are likely to be affected?
- When a component is changed, what requirements need to be retested?
- Impact analysis starts in development by programmers and continues during system testing

Impact Analysis

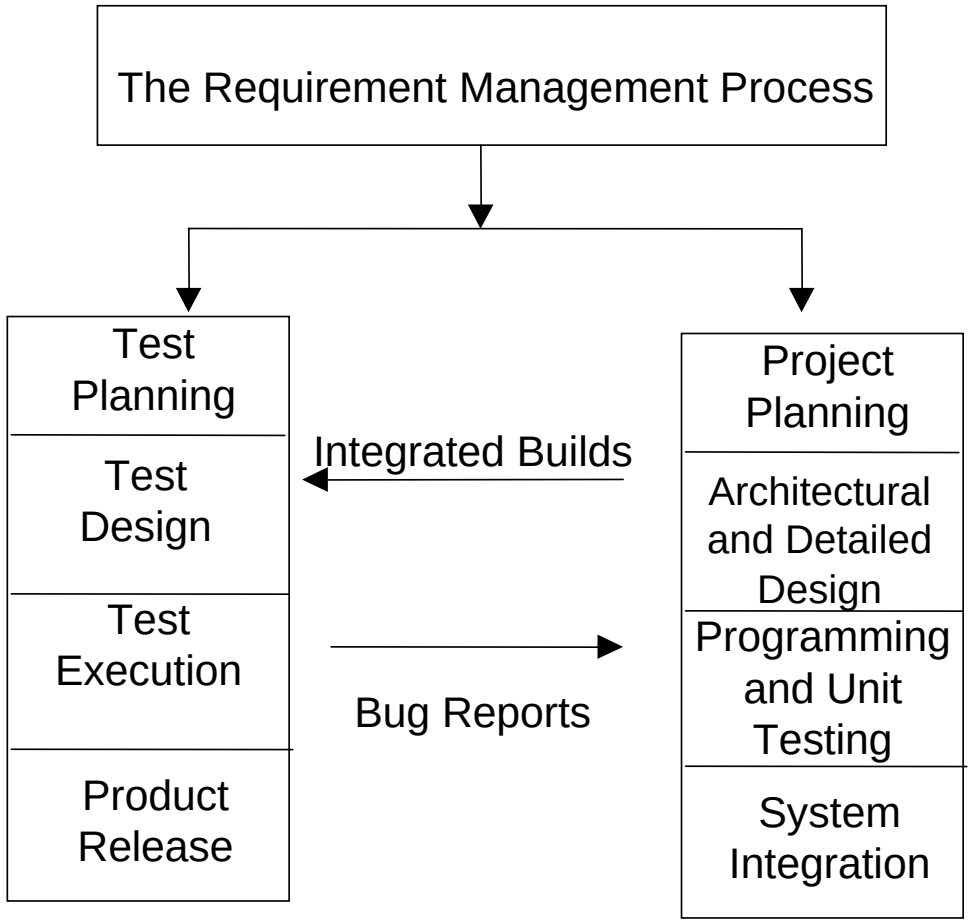
	Req 1	Req 2	Req 3	Req 4	Req n
Component 1	X		X		
Component 2			X	X	
Component 3	X	X	X		
Component 4					
Component n					

Practice #9
Define Testing As a Process NOT As a Lifecycle Phase



????

The Common Sense Model for Software Development and Testing



Test Planning Activities

- Specify test environment
- Specify scope and objectives
- Specify your approach to systems test
- Perform risk analysis
- Determine staffing requirements and responsibilities
- Determine any hardware, software, and network resources required for systems test
- Decide on tools to be used and determine training needs
- Define tasks and their sequence

Test Design Activities

- Define test scenarios and test cases
- Identify and creating test data
- Determine expected results for each test
- Define test procedures to conduct each test
- Determine the test cycles and their sequence
- Write any programs or command control language needed to download data to the test environment
- Set up the test environment
- Write and test any programs or command control language needed to compare expected results with actual results
- Write and test any programs or command control language needed to conduct the tests

Practice # 10
Select Tools to Support Your Test Process

- Test planning tools
- Test management tools
- Test case design tools
- Test coverage tools
- Test execution and record/playback tools
- Static analysis tools
- Defect tracking tools

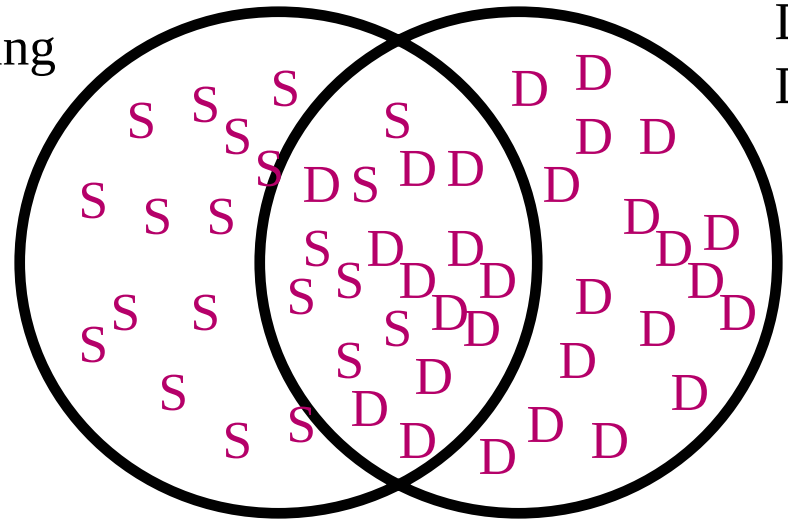
Practice #11
Perform Both Dynamic and Static Testing

- Dynamic testing is the process of executing a program or system with the intent of finding errors or making sure that the system meets its intended requirements

- Static testing is any activity that aims at finding defects by inspecting, reviewing and analyzing any static form of the software (requirement models, design models, code, test plans and test design specification)

Balance Between Dynamic and Static Testing

S: Defects found during Static Testing



D: Defects found during Dynamic Testing

Many defects that are normally left for dynamic testing can be found early through static testing

Practice #12
Continuing Formal Education

- Testing SW is an engineering discipline
- On-the-job training can only go so far
- Tester Certification is one way
  - Certified Software Testing Professional (CSTP)
    - Covers the areas a well-rounded tester must know
    - Develop a sense of professionalism
    - Help change the perception of testers as non-professionals
    - Meet tomorrow's challenges
  - See the body of knowledge document in appendix

CSTP Body of Knowledge

1. **Principles of Software Testing**
   - Levels of Testing
   - Testing client/server applications
   - Testing Internet and web applications
   - Testing object-oriented applications
   - Testing embedded systems
   - The testing life cycle

2. **Test Design**
   - Code-based test case design techniques
   - Requirement-based test case design techniques
   - Test design specification

CSTP Body of Knowledge

3. Managing the Testing Process

- Planning
- Scheduling
- Reporting
- Resources
- Risk Management
- Measuring and improving the test process

CSTP Body of Knowledge

4. **Test Execution and defect tracking**
   - Test scripting
   - Reporting
   - Defect tracking

5. **Requirement Definitions, Refinement and Verification**
   - Writing testable requirements
   - Exploring requirements
   - Refining requirements
   - Defining requirements
   - Requirement verification
   - Requirement tractability

CSTP Body of Knowledge

6. Test automation

- Tool evaluation and selection
- Architectures
- Automation standards and guidelines
- Planning the test automation process
- Automation team roles

7. Static Testing (Inspections, Reviews, and Walkthroughs)

- Types of static testing
- The process of static testing
- Defect data analysis
- Improving the process