Unit 5. Testing Concepts and Management

1. Introduction
2. Testing Dimensions
3. Test Concepts
4. Testing Process
5. Test Documentation

Reading: TB-Chapters 1 (1.3-1.15), 3 (3.4)

1. Introduction

-Testing is the process of ***uncovering evidence of flaws and fixing these flaws*** in software systems.

- Flaws may result from various reasons such as ***mistakes***, ***misunderstandings***, and ***omissions*** occurring during any phase of software development.

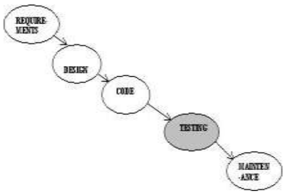
-Testing allows for ***mitigating software risks***.

- ***Testing*** is conducted according to a ***test plan***, which is more effectively driven by the specific risks faced by the system.
- The ***risks*** determine ***what type of tests*** need to be conducted;
- The ***test plan*** determines ***how much testing*** is needed or is acceptable from a business perspective.

-Testing remains one of the most costly and challenging aspects of the software development process.

- From a business perspective, there is an ***optimum level of testing***, which is acceptable.
- Beyond the optimum level, testing becomes less cost-effective, because the cost of testing simply exceeds the gains obtained from the defects detected.

Lifecycle Testing



-Traditionally testing used to occur at the latter phases of the software cycle

-Studies conducted by IBM over several major projects have shown that:

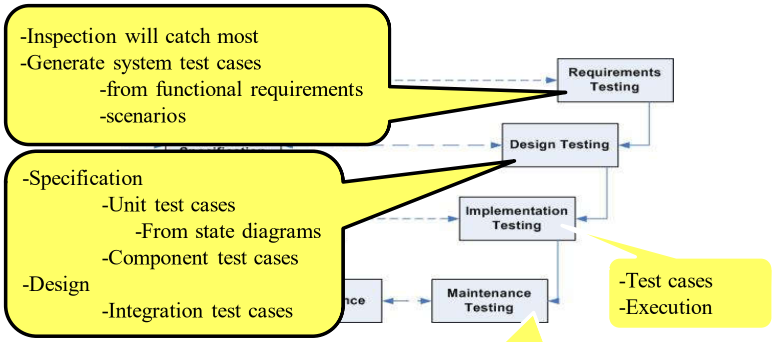
- An average of 60 defects could be detected during application development:
  - 50% of the defects could be detected when testing is conducted before the implementation.
  - 80% could be detected when testing is conducted after implementation.
  - It is at least 10 times more expensive to fix a defect after implementation than before, and 100 times more expensive during maintenance.

-Lessons learned:

- Importance of starting testing as early as possible.
- Necessity to integrate testing in the entire development life cycle, and not only after coding or during production.

-*Lifecycle testing:* recommended as alternative to traditional testing

- Incorporates testing in all the phases of the development process, and as such it spans the entire software lifecycle.
- Starts at the same time as the product development process; both processes run concurrently and should be conducted using well structured but different methodologies.



-Inspection will catch most
-Generate system test cases
-from functional requirements
-scenarios

-Specification
-Unit test cases
-From state diagrams
-Component test cases
-Design
-Integration test cases

-Test cases
-Execution

-Maintenance-based Testing

- *Requirements Testing:* ensure that the requirements orded or documented properly, a s.
- *Design Testing:* ensure that t matches with the requirements.
- *Implementation Testing:* ensure that the design specification has been correctly implemented.
- *Maintenance Testing:* involve deployment testing at the installation, and regression testing during operation.

2. Test Dimensions

-Since program errors are diverse and complex, no single test technique can cover all kinds of errors.

- In practice, a combination of several techniques is used to achieve adequate testing.
- There are many levels of testing as well as many dimensions to testing, which should be considered in order to test appropriately a software system.

Test Scopes

-There are three levels of testing:

- ***Unit Testing***: targets small piece of software entity such as a method, a class, a cluster of interdependent classes, or a software component.
- ***Integration Testing***: tests the interactions between a collection of related functions or components.
- ***System Testing***: test the whole system after all the components or subsystems are combined into the final product.

Testing Approaches

-Based on the test data available, testing can be categorized in either functional testing or structural testing.

-***Functional testing***, also called black box testing, consists of checking the system without having or using any knowledge of the internal logic in developing the test cases.

- Checks that the software system is built according to the requirements.

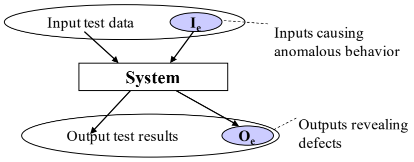
-***Structural testing***, also called white-box testing, uses knowledge of the internal logic to develop test cases.

- Checks the structure of the software artifacts (e.g., code, design etc.).

3. Test Concepts

-*Test case*: specification of inputs and outputs required to reveal defects

- Consists of the pretest state and environment, the test inputs and **expected outputs** and state.
- Expected outputs may include data, messages or exceptions generated by the application under testing.



-*Test oracle*: a technique or mechanism used to produce expected results under specific conditions.

-*Test strategy*: an algorithm and/or a collection of heuristics used to identify interesting test cases for an application under testing.

-*Test suite*: a collection of test cases, typically related by a testing goal or implementation dependency.

Example: an IUT and some corresponding test cases and test suite

```
void prog(int X, int Y) {
            x = X; y = Y;
            if (x > 100) {
                        x = x – 100;
            } else {
                        x = x + 100;
            }

            if (x <= y) {
                        y = 2*y;
            }
            else {
                        if (x > 200) {
                                    x = x - 100;
                        } else {
                                    x = x*x;
                        }
            }
            printf("x=%d. y=%d",x, y);
}
```

Sample test cases:

Test case Id	Input		Expected Output	
	X	Y	x	y
Tc1	10	100	110	200
Tc2	101	100	1	100
Tc3	201	110	1	110

Sample test suite: {Tc1, Tc2, Tc3}

How where the test cases generated?

Test concepts (ctd.)

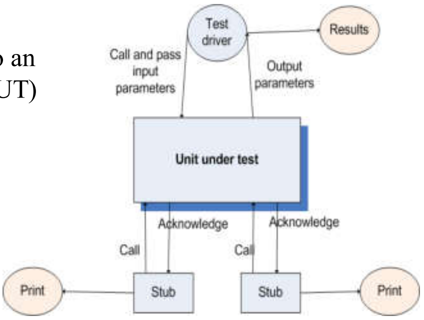
-*Test run:* the execution with actual results of a test suite(s).

-Actual results are compared against expected ones, in order to decide the outcome: ***pass*** or ***no pass***.

- ***pass:*** actual results are the same as expected ones.
- ***no pass:*** actual results are different from expected ones; this reveals a bug and is therefore considered a ***successful*** test.

-*Test driver*: a class or utility program that applies test cases to an implementation under testing (IUT)

-*Stub:* a partial, temporary implementation of a component, that may serve as a placeholder for an incomplete component or implement testing support code.



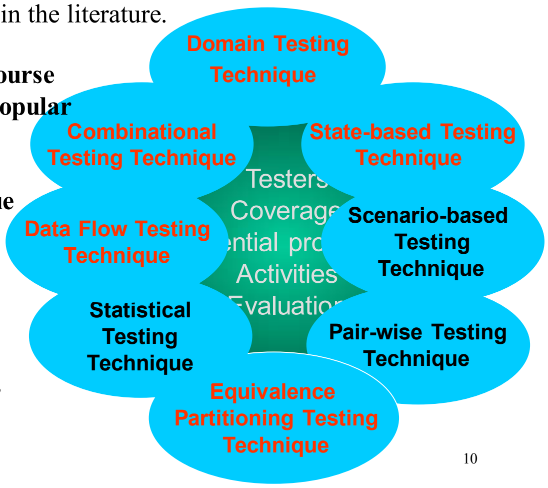
Test Strategies

-More than 200 testing approaches have been published in the literature.

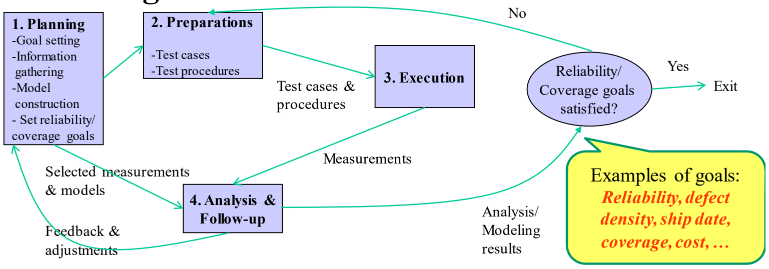
-We'll study in this course many examples of popular testing techniques.

"So Which Technique Is the Best?"

- Each has strengths and weaknesses
- Think in terms of complement
- There is no "one true way"
- Mixing techniques can improve coverage


Domain Testing Technique
Combinational Testing Technique
State-based Testing Technique
Testers
Coverage
ential pro
Activities
Evaluatio
Data Flow Testing Technique
Scenario-based Testing Technique
Statistical Testing Technique
Pair-wise Testing Technique
Equivalence Partitioning Testing Technique

4. Testing Process



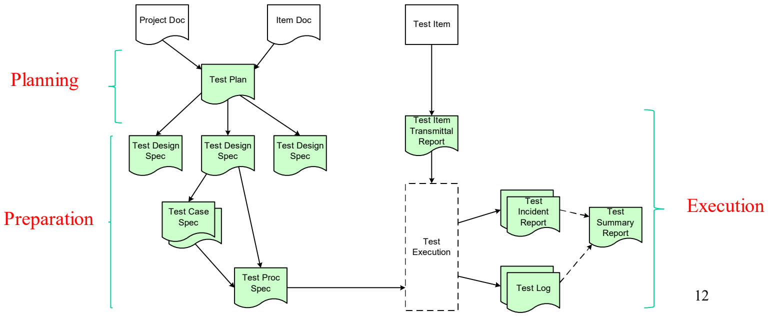
Examples of goals: ***Reliability, defect density, ship date, coverage, cost, …***

- The major test activities include the following:

1. **Test Planning:** set the goals for testing, and select an overall testing strategy, adequate quality metrics and models.
2. **Test Preparation:** prepare specific test cases and the general test procedure.
3. **Test execution:** execute test cases and collect measurement of product behavior.
4. **Analysis and follow-up:** include result checking and analysis to determine if a failure has been observed, and if so, follow-up activities are initiated to remove the underlying causes.

5. Test Documentation

- IEEE Standard 829 for software test documentation is a standard initially published by the IEEE in 1983 and later approved by the American National Standards Institute (ANSI).
- The standard describes a wide range of information that can be included in test documentation.
- The IEEE standard 829 defines the following test document templates (in green):



Test Case Specification

A test case specification shall have the following structure:

- Test case specification identifier
- Test items
- Input specifications
- Output specifications
- Environmental needs
- Special procedural requirements
- Intercase dependencies

Test Case Template

Build Number	
Tester Name	
Test Type	
Test Case Name	
Test Case Number	
Test Case Description	
Items to be tested	
1	
…	
n	
Specifications	
Input	**Expected Output**
Procedural Steps	
1	
…	
m	

Sample Test Case:

Build Number	P6_306
Tester Name	John Abott
Test Type	Unit Test
Test Case Name	testGetAuthor
Test Case Number	TC0014
Test Case Description	Testing that the ***getAuthor*** method of class ***Patent.java*** retrieves the patent authors' names for a patent document.
Items to be tested	
1	Check that the method returns null if the name is not set yet
2	Check that one can set the authors names with setAuthor() and then retrieve it with getAuthor()
Specifications	
Input	**Expected Output**
void	Author's name as a string which is entered with the setAuthor(author_name) function, or null if name not set yet
Procedural Steps	
1. Create a new Patent.java instance 2. Call method getAuthor() and ensure that the return value is null 3. Call setAuthor(name_of_author) followed by getAuthor() and ensure the names of the authors are the same	

Test Planning

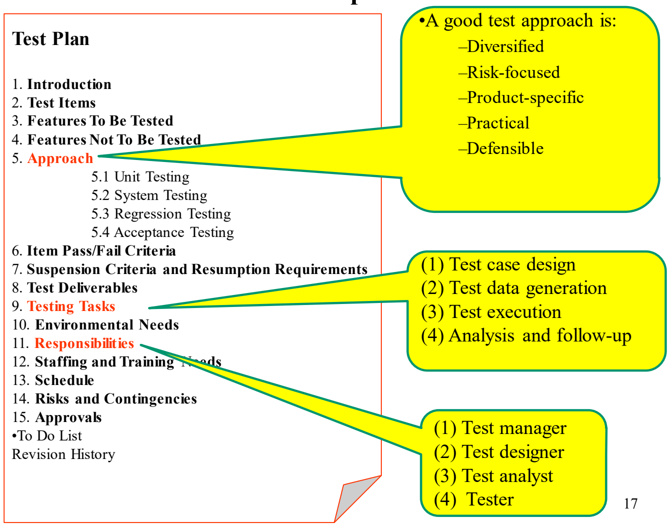
-Start at the ***beginning of the development process***

- Represent the first task in any testing process

-Define the roadmap to be followed during the testing process.

- Provide background information on the system under testing
- Specify the test objectives
- Identify and describe the test and risk factors involved
- Provide a description of the components and functions to be tested, and the nature of the tests to be conducted
- Identify the potential members of the test team and their responsibilities.
- Identify test phases, and specify for each test phase, the tasks to be done and the workers assigned
- Identify test resources, environments, and tools
- Establish preliminary test schedule and estimate initial test costs

IEEE std 829-1983 Test Plan Template



Test Plan

1. **Introduction**
2. **Test Items**
3. **Features To Be Tested**
4. **Features Not To Be Tested**
5. **Approach**
   - 5.1 Unit Testing
   - 5.2 System Testing
   - 5.3 Regression Testing
   - 5.4 Acceptance Testing
6. **Item Pass/Fail Criteria**
7. **Suspension Criteria and Resumption Requirements**
8. **Test Deliverables**
9. **Testing Tasks**
10. **Environmental Needs**
11. **Responsibilities**
12. **Staffing and Training Needs**
13. **Schedule**
14. **Risks and Contingencies**
15. **Approvals**

- To Do List

Revision History

- A good test approach is:
  - –Diversified
  - –Risk-focused
  - –Product-specific
  - –Practical
  - –Defensible

(1) Test case design
(2) Test data generation
(3) Test execution
(4) Analysis and follow-up

(1) Test manager
(2) Test designer
(3) Test analyst
(4) Tester

Test Status Reporting

-Addresses issues that are of high interest to management such as:

- When the final product will be ***released***,
- Whether or not ***enough testing*** has been achieved,
- How ***reliable*** will the system be.

-Since the test (status) report serves as a management decision-making tool, it should be ***short, concise, and easy to read***.

The Overall Structure of a Status Report

- Here’s one structure that some managers find works well for them:
  - The report has four parts, each part starts a separate page.
  - Part 1 Risks and responsibilities
  - Part 2 Progress against plan
  - Part 3 Project bug metrics
  - Part 4 Deferred and no-fix bugs to approve

•Part 1: Risks and responsibilities

- Highlights current problems, such as:
  - Artifacts due into testing but ***not arrived***
  - Artifacts that due out of testing but ***not yet completed***
  - Staff turnover that ***threatens the schedule***
  - Equipment acquisition problems that might threaten the schedule.

The Overall Structure of a Common Report (ctd.)

•Part 2: Progress against plan

Can be documented in the following table:

Component	Test Type	Tester	Total tests Planned/ Created	Test passed/ failed	Time budget	Time spent	Projected effort for next build	Notes
Frame Processor	Unit	J. Carrey	297	264/33	2 weeks	3 weeks	...	...
...								
...								

- Note how this covers progress against a plan, risks/obstacles, effort and results, all in one chart

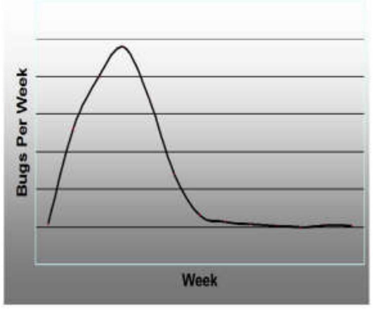
The Overall Structure of a Common Report (ctd.)

•Part 3: Project bug metrics

- These charts show ***find / fix rates*** over the course of the project.
- Useful to give a sense of the ***rate at which problems are being repaired***.
- If the repair rate near the end of the project is slow compared to the find rate, the ***schedule is at risk***.

- Example- Bug Counts and Extent of Testing:

Attempt to measure testing progress by plotting a project’s bug numbers against a theoretical curve of expected find rates over time.



–***Caution:*** It could be too easy to over-interpret these charts

The Overall Structure of a Common Report (ctd.)

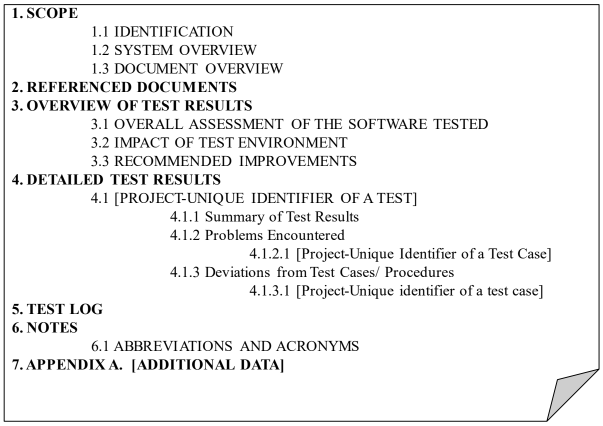
•Part 4: Deferred and no-change change requests

- Every project team fixes some bugs and rejects or defers others.
  - At some point, there must be management review of the collection of problems that will not be fixed.
- Rather than save up the list for the end of the project, list the new not-to-be-fixed change requests every week.

Software Test Report (STR): Template *(MIL-STD-498 DID, DI-IPSC-81440)*

-Test status reports are produced regularly during the test process.

-At the end of the process, an overall test report summarizing and analyzing the results should be produced as well.



1. SCOPE
1.1 IDENTIFICATION
1.2 SYSTEM OVERVIEW
1.3 DOCUMENT OVERVIEW
2. REFERENCED DOCUMENTS
3. OVERVIEW OF TEST RESULTS
3.1 OVERALL ASSESSMENT OF THE SOFTWARE TESTED
3.2 IMPACT OF TEST ENVIRONMENT
3.3 RECOMMENDED IMPROVEMENTS
4. DETAILED TEST RESULTS
4.1 [PROJECT-UNIQUE IDENTIFIER OF A TEST]
4.1.1 Summary of Test Results
4.1.2 Problems Encountered
4.1.2.1 [Project-Unique Identifier of a Test Case]
4.1.3 Deviations from Test Cases/ Procedures
4.1.3.1 [Project-Unique identifier of a test case]
5. TEST LOG
6. NOTES
6.1 ABBREVIATIONS AND ACRONYMS
7. APPENDIX A. [ADDITIONAL DATA]