



**LOGIGEAR CERTIFIED TEST  
PROFESSIONAL I**

# **BASIC SOFTWARE TESTING SKILLS**

- Chapter 1: Software Development Life Cycles and Testing
- Chapter 2: Software Testing Overview
- Chapter 3: Test Requirements
- Chapter 4: Test Design Techniques
- Chapter 5: Working with a Test Case Management System
- Chapter 6: Software Error
- Chapter 7: Working with a Bug Management System
- Chapter 8: Reporting Progress: Status Report



## **CHAPTER 1**

# **Software Development Lifecycles (SDLC) and Testing**

- SDLC and Testing
- Waterfall Model
- Spiral Model
- V-Model
- Concurrent Model
- Agile Model
- Other SDLC Models
- Testing Phases and Milestones

# OBJECTIVES

- An introduction to Common Software Development Life Cycle (SDLC) practices and how software testing fits in those contexts.
- Understanding testing phases, milestones, checkpoints in a SDLC and their purposes
- Learn about how each SDLC choice affects software testing and its implementation

# **SDLC and Testing**

## **1.1 SDLC and Testing**

1.2 Waterfall Model

1.3 Spiral Model

1.4 V-Model

1.5 Concurrent Model

1.6 Agile Model

1.7 Other SDLC Models

1.8 Testing Phases and Milestones

# **SDLC and Testing**

## **How testing is directly affected by the SDLC choice:**

- Documentation availability to test against
- Time to test
- Time to automate or produce effective automation
- Knowledge and understanding of the application as we plan our tests
- Amount of regression testing

# SDLC and Testing

1.1 SDLC and Testing

**1.2 Waterfall Model**

1.3 Spiral Model

1.4 V-Model

1.5 Concurrent Model

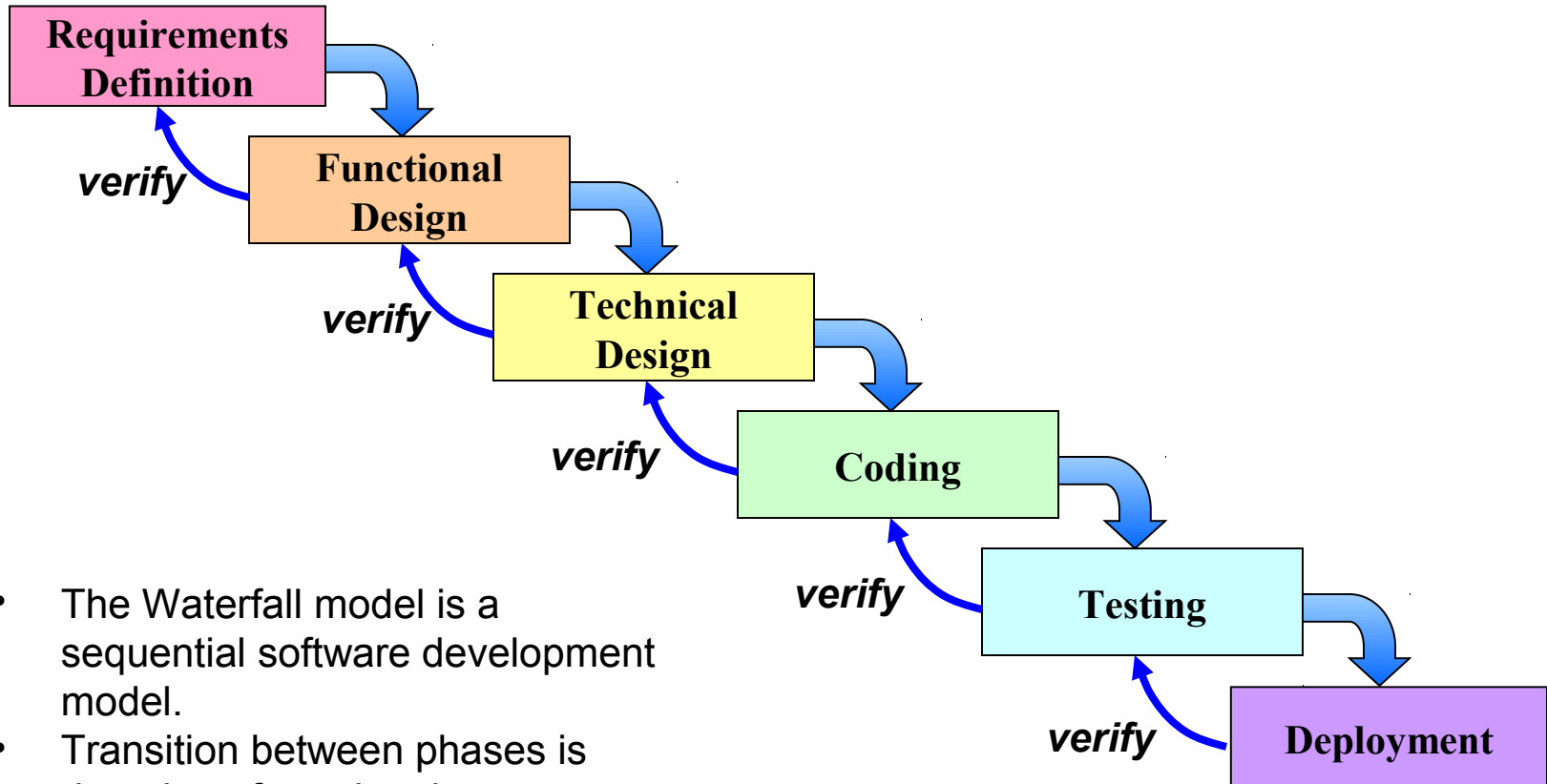
1.6 Agile Model

1.7 Other SDLC Models

1.8 Testing Phases and Milestones



## CHAPTER 1.2

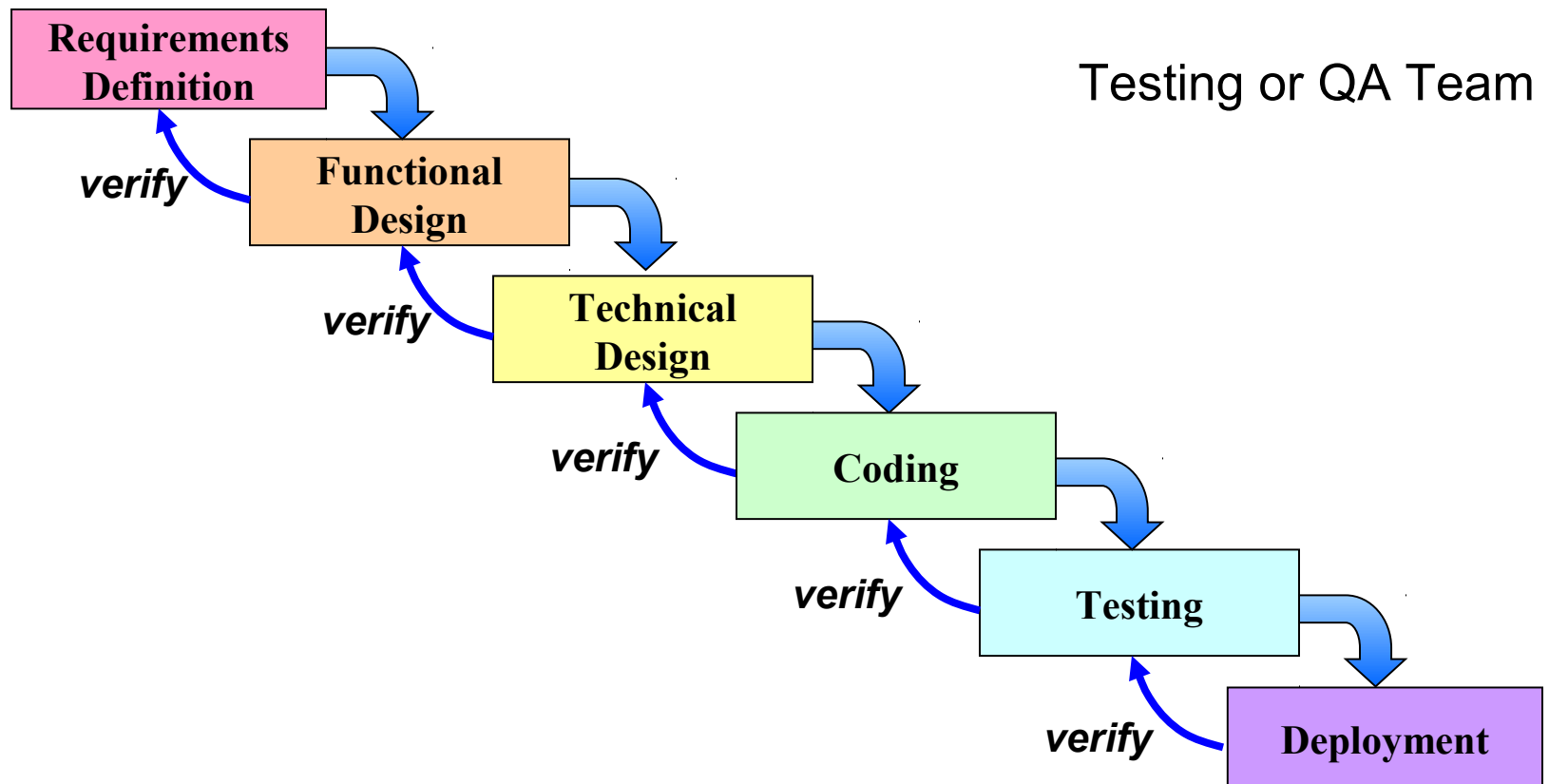


- The Waterfall model is a sequential software development model.
- Transition between phases is done by a formal review.
- The review is a checkpoint to see that you are on the right track.

# Testing in Waterfall Model

- Testing is not inherent to every phase of the Waterfall model.
- Constant testing from the design, implementation and verification phases is required to validate the phases preceding them.
- **Testing phase:** Only after coding phase, software testing begins. Different testing methods are available to detect the bugs that were committed during the previous phases. A number of testing tools and methods are already available for testing purposes.

# Roles of Testing Team in Waterfall Model



# SDLC and Testing

1.1 SDLC and Testing

1.2 Waterfall Model

**1.3 Spiral Model**

1.4 V-Model

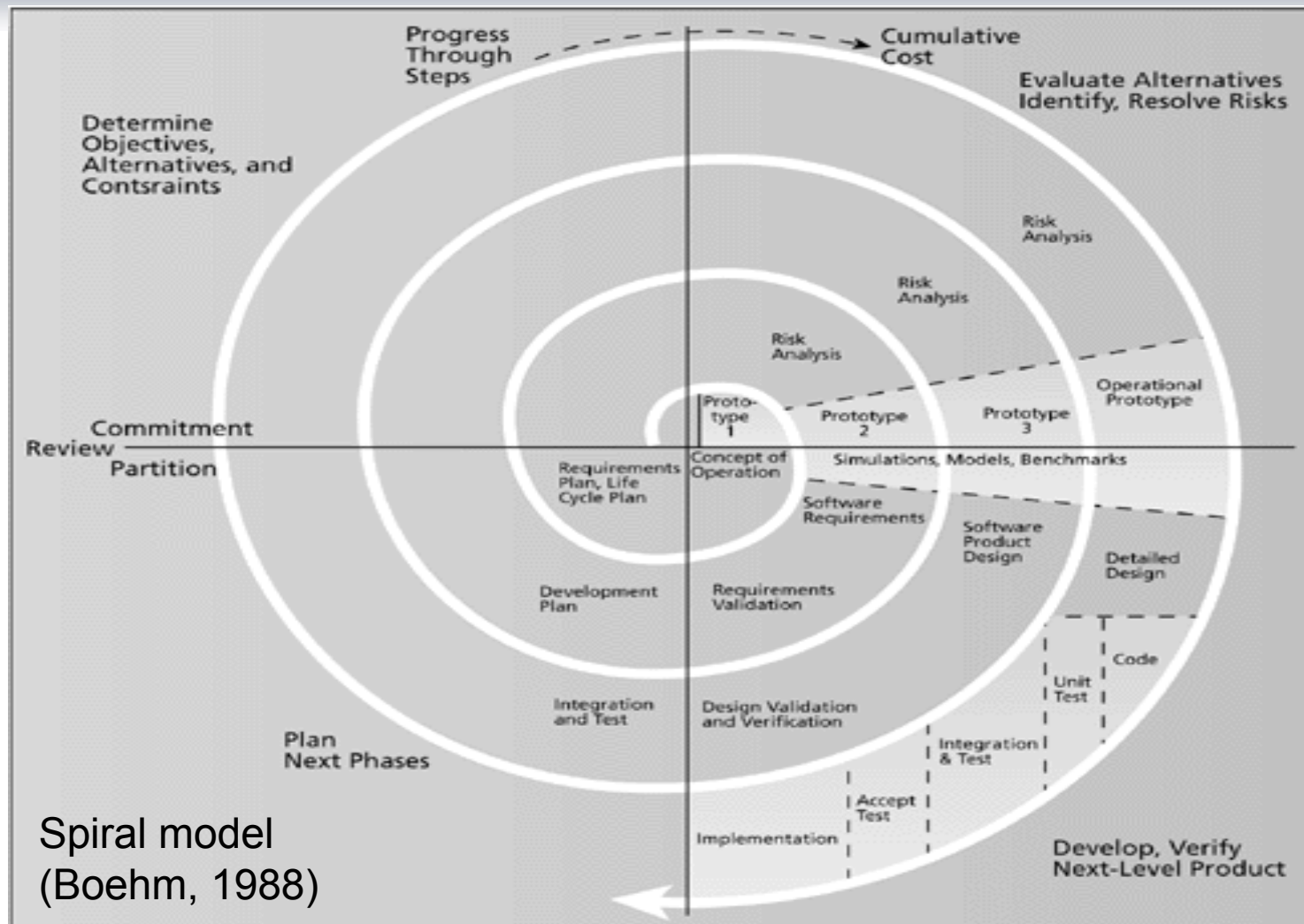
1.5 Concurrent Model

1.6 Agile

1.7 Other SDLC Models

1.8 Testing Phases and Milestones

# Spiral Model



# **Testing in Spiral Model**

- Planned and structured releases
- Usually has documentation to test against
- Each spiral iteration can be thought of as a “mini-waterfall”; there are defined testing phases
- Previous releases must be regression tested

# **SDLC and Testing**

1.1 SDLC and Testing

1.2 Waterfall Model

1.3 Spiral Model

**1.4 V-Model**

1.5 Concurrent Model

1.6 Agile Model

1.7 Other SDLC Models

1.8 Testing Phases and Milestones

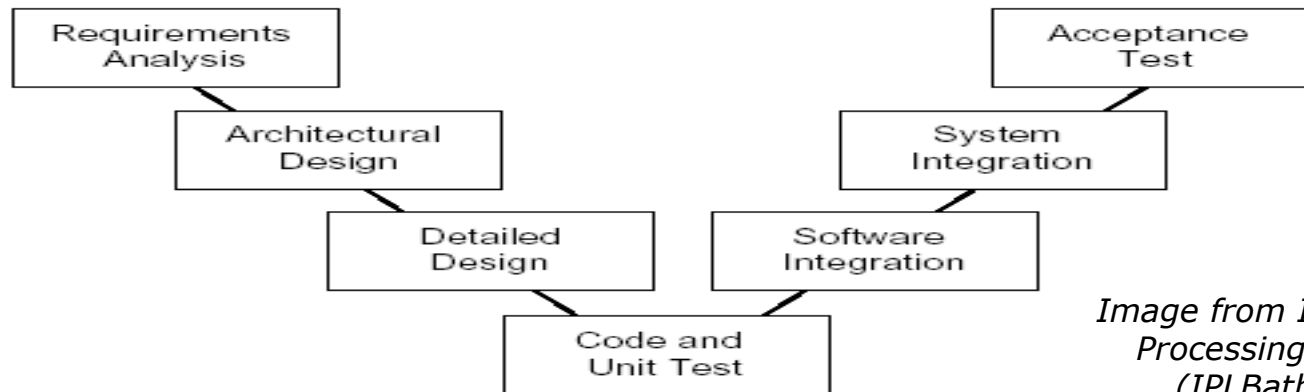
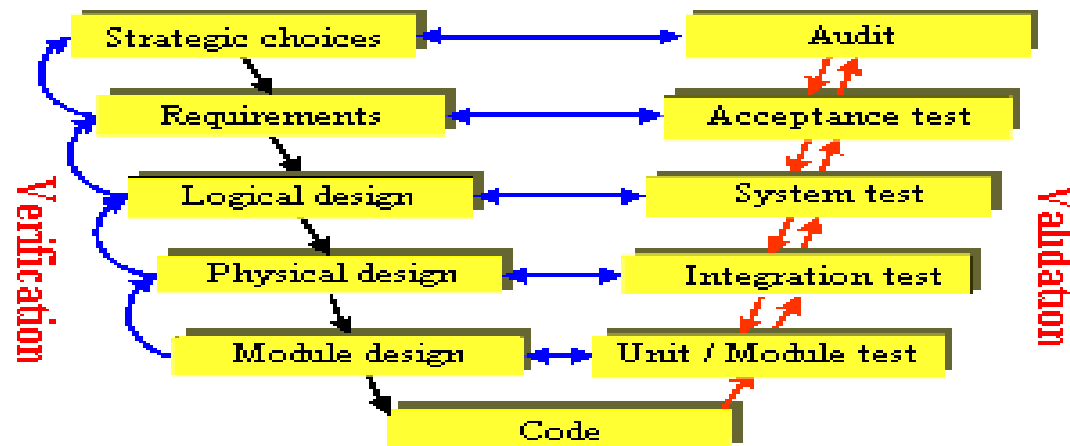


Image from Information Processing Limited (IPLBath.com)

## The V-model



Source: Glenford Myers (1979)



## CHAPTER 1.4

- Verification: Checks that a deliverable is complete (Contains all requires information, follows standards; verify a procedure).
- Validation: Checks that the deliverables satisfy requirements specified in the previous stage or an earlier stage, and that the business case is met (Validate a function or requirement).
- Testing: Ensures that the specification is properly assembled and implemented (Test to see if it works).
- In testing, these are important changes in software development from the V-model:
  - Write the test cases at requirements review.
  - Unit testing

# **SDLC and Testing**

1.1 SDLC and Testing

1.2 Waterfall Model

1.3 Spiral Model

1.4 V-Model

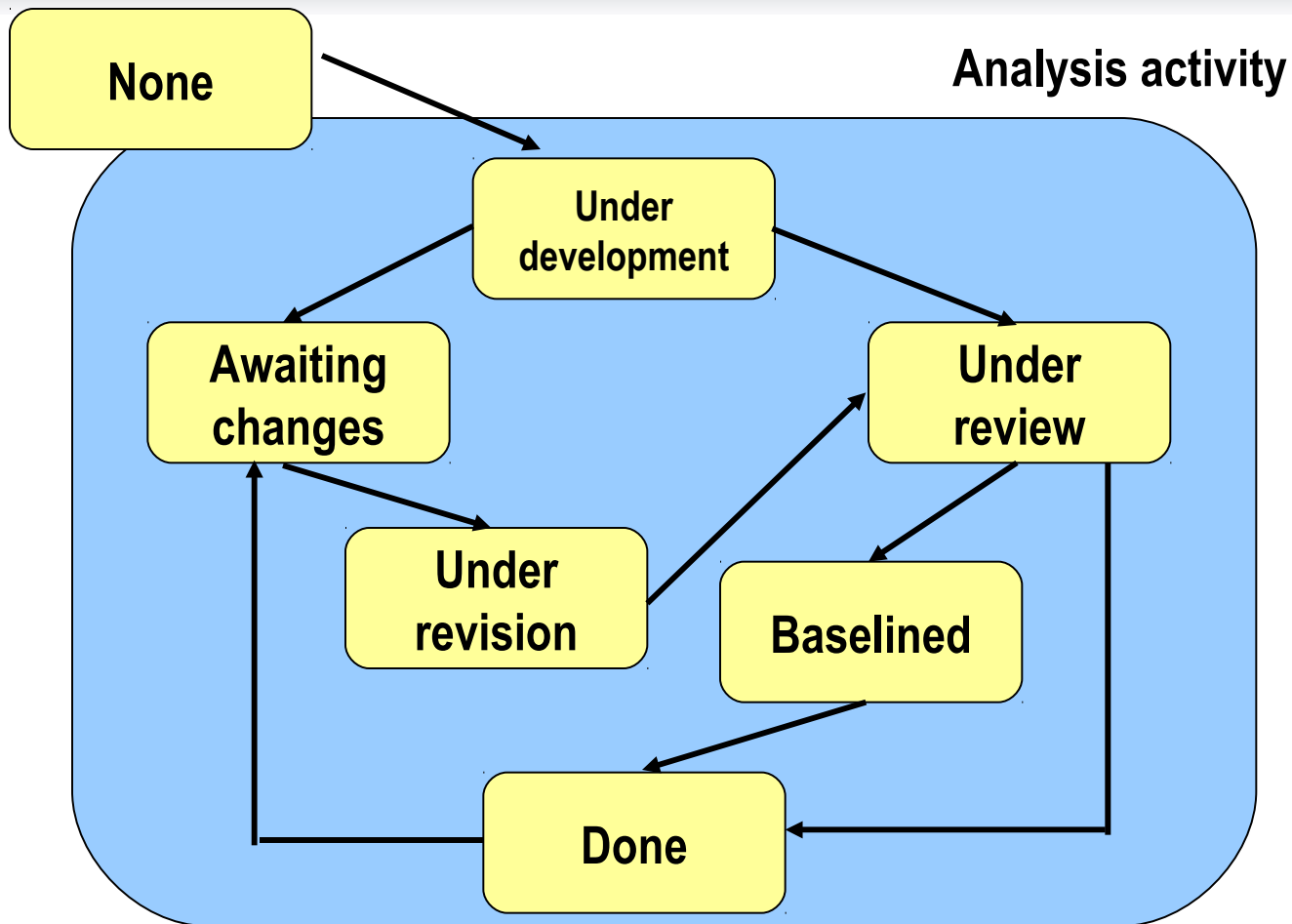
**1.5 Concurrent Model**

1.6 Agile Model

1.7 Other SDLC Models

1.8 Testing Phases and Milestones

# Concurrent Model



All activities concurrent but reside in different states

# Testing in Concurrent Model

- Planning is concurrent to design; design is concurrent to development—everything is happening at the same time.
- The whole project is not well planned or well structured.
- Planning, design and development are most dynamic. Product is in constant change. Very difficult to test; impossible to effectively plan testing project.
- Often no documentation to test against.
- Testing is ad hoc.
- Coverage usually cannot be measured. Structured regression testing is impossible.
- Bugs will be missed because of so much change and so little planning.
- Risk analysis and reporting is crucial.

# **SDLC and Testing**

1.1 SDLC and Testing

1.2 Waterfall Model

1.3 Spiral Model

1.4 V-Model

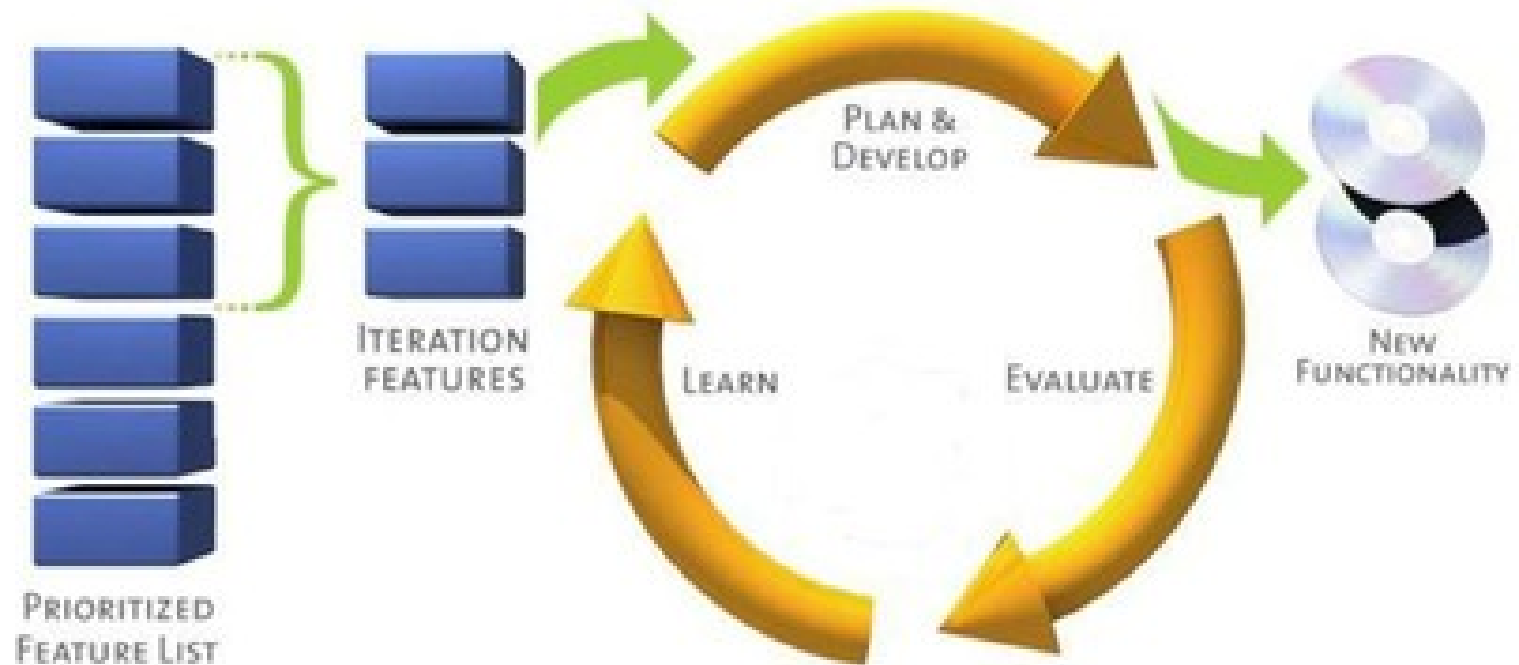
1.5 Concurrent Model

**1.6 Agile Model**

1.7 Other SDLC Models

1.8 Testing Phases and Milestones

# Agile Model



**Source: No Glory in “The Middle Way“**  
(The “Balanced Blend” Manifesto takes Shape)  
**March 21, 2008**  
by Mike Cottmeyer

# Testing in Agile Model

- Delivery cycles are short. Development is very focused. Diagrams, use cases, user stories, index cards, or discussions of functionality serve as “documentation” to test against.
- Agile projects usually have more unit testing by developers.
- Dynamic nature of development needs structured regression testing.
- Testing is often ad hoc, but focused.
- Dynamic style or projects include much change but the change is discussed and side-effects noted.

# **SDLC and Testing**

1.1 SDLC and Testing

1.2 Waterfall Model

1.3 Spiral Model

1.4 V-Model

1.5 Concurrent Model

1.6 Agile Model

**1.7 Other SDLC Models**

1.8 Testing Phases and Milestones



# Other SDLC Models

- Rapid Application Development (RAD) Model
- Test First
- Rational Unified Process (RUP)
- eXtreme Programming (XP)
- ...the list goes on

# **SDLC and Testing**

- 1.1 SDLC and Testing
- 1.2 Waterfall Model
- 1.3 Spiral Model
- 1.4 V-Model
- 1.5 Concurrent Model
- 1.6 Agile Model
- 1.7 Other SDLC Models
- 1.8 Testing Phases and Milestones**

# Phases & Milestones

- Development Phase: A time block within SDLC. There are a number of activities that have to be done in this time block.
- Examples of some Test Phases: Unit Test, Integration Test, System Test, User Acceptance Test, Release Test and Maintenance Test.
- Development Milestone: A significant event in the software developing process in which, the software moves from one phase to another. The milestone let us know where the application is in SDLC
- Criteria: Built or created from the standard of the project.

# **Product Development Milestones**

## **Examples**

- Product Design and Specification Complete
- First Functionality
- Pre-Alpha or Alpha Candidate
- Alpha
- Pre-Beta or Beta Candidate
- Beta
- User Interface Freeze
- Pre-Final or Golden Master Candidate (GMC)
- Final Test
- Release or Golden Master

# Testing Phases and Milestones

Test Phase	Pre – Alpha Unit	Alpha Integration	Beta System	GM/Release User Acceptance
Performed by	Developers	Developers/ (Technical) Testers	Testers	Users
Definition	Test one unit of code at a time	Iteratively more complex system components integrated. Checking the units of code work together incrementally	Big bang. The entire product as it is intended to be used	Requirements-based testing by the project sponsor, testing the production system
Outcome	Fewer Bugs	Bug Finding	Bug Finding/QC	QC
Goals of Testing	To validate each module in isolation	To find bugs, to learn about the product, to validate the integration requirements	To find bugs, to validate the system requirements, data flow, scenarios, load	To validate user requirements

## Entry/Exit Milestone Criteria

Example Tasks and Activities for the Test Team:

- Test Plan reviewed
- Test Plan finalized
- Requirements reviewed and base-lined
- Test system approved and built
- Bug database opened and available for bug entry
- ...

# **Example**

## **Alpha Phase Entry**

- The Test Plan covering all test phases:
  - Source Level Testing, Application Testing, and Integration Testing are complete and approved.
  - Developer's testing sign-off document is complete
  - Approved Unit Test Checklists are available for review
  - Test Design/ Cases are complete and approved
  - Build acceptance test is executed successfully
  - Regression test has been defined
  - Automation test has been defined (if applicable)
  - The hardware and software necessary for the tests, as identified in the Test Plan, are available and ready

# **Example**

## **Alpha Phase Exit**

- All test results have been documented
- All identified application testing has been executed
- Regression test has been executed
- Automation test has been executed (if applicable)
- All software fixes identified for resolution in this test phase are fixed, retested and closed
- Software is baselined
- Test report is completed
- All defects have been resolved and closed



- Depending on the type of application to be built and or customer's requirement, we choose a suitable SDLC model.
- There are several phases in the SDLC. In each phase, there are milestones and criteria for each milestone. The criteria, milestones and phases of the project help the product to be developed correctly, timely and effectively.
- Depending on the SDLC, the testing and its implementation is affected, usually in time, knowledge about the application, documentation and amount of regression test done.



## CHAPTER 2

# Software Testing Overview

- Testing Group
- Objectives of Testing
- Test Planning Framework
- Test Coverage
- Manual Testing vs. Automated Testing
- Key Lessons-Learned

# OBJECTIVES

- An introduction to the concept of quality assurance (QA), quality control (QC), and software testing
- The roles of software testing groups
- Objectives of testing and some testing issues
- An introduction to Manual Testing and Automated Testing
- An introduction to LogiGear© Software Test Planning Framework and possible test strategies, approaches, methods, types and techniques used in various software application development phases, for specific quality objectives

# **An Overview of Testing**

## **2.1 Testing Group**

2.2 Objectives of Testing

2.3 Test Planning Framework

2.4 Test Coverage

2.5 Manual Testing vs. Automated Testing

2.6 Key Lessons-Learned

# Testing, QA, QC

- **Quality Assurance:** A process for providing adequate assurance that the software products and processes in the product life cycle conform to their specific requirements and adhere to their established plans.
- **Quality Control:** A set of activities designed to evaluate a developed working product.
- **Testing:** The process of executing a system with the intent of finding defects including test planning prior to the execution of the test cases.

The key difference to remember is that

*Quality Assurance is interested in the process  
whereas*

*Testing and Quality Control are interested in the product.*

**Having a testing component in your development process demonstrates a higher degree of quality (as in QA).**

# **Roles of the Testing Group**

- The Testing Group provides important technical services, including:
  - Finding and reporting bugs.
  - Identifying weak areas of the program.
  - Identifying high risk areas of a project.
  - Explaining findings in ways that help
    - Engineering solve or fix the problems.
    - The customer service staff help customers.
    - Management make sound business decisions about each bug.

# Typical Responsibilities of a Tester

- Find Problems
  - Find bugs.
  - Find design issues.
  - Find more efficient ways to find bugs.
- Communicate Problems
  - Report bugs and design issues.
  - Report on testing progress.
  - Evaluate and report the program's stability.
- More Senior Testers Manage/Supervise Testing Projects
  - Prepare test plans and schedules.
  - Estimate testing tasks, resources, time and budget.
  - Measure and report testing progress against milestones.
  - Teach other testers to find bugs.



# An Overview of Testing

2.1 Testing Group

**2.2 Objectives of Testing**

2.3 Test Planning Framework

2.4 Test Coverage

2.5 Manual Testing vs. Automated Testing

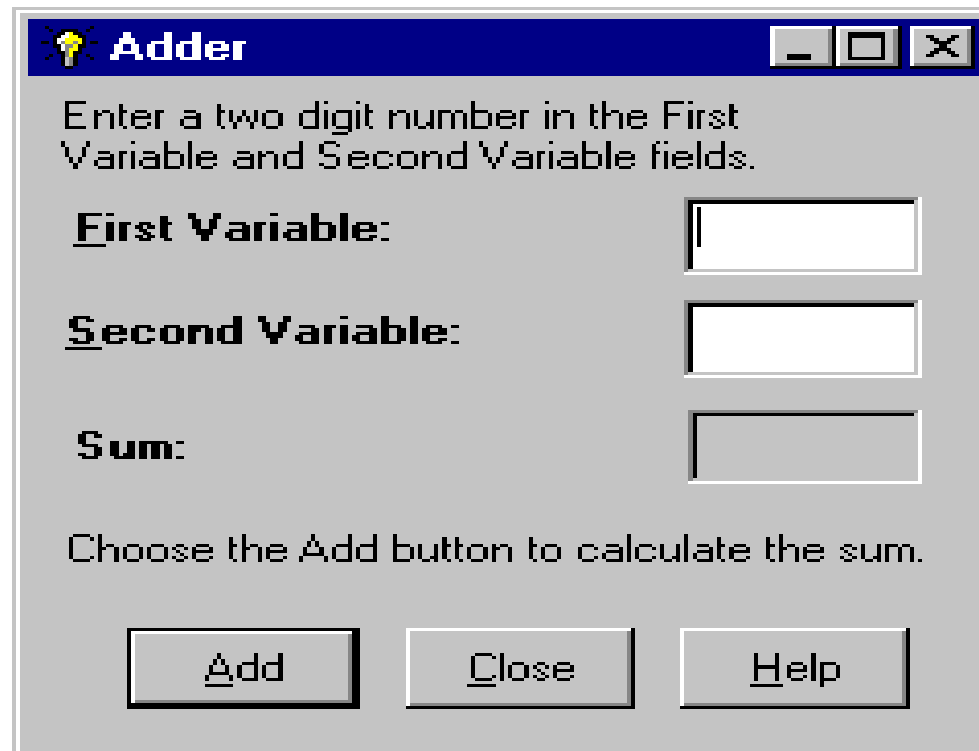
2.6 Key Lessons-Learned

# **Example Test Series**

The program's specification:

- This program is designed to add two numbers, which you will enter
- Each number should be one or two digits

# Example Test Series



**Adder**

Enter a two digit number in the First Variable and Second Variable fields.

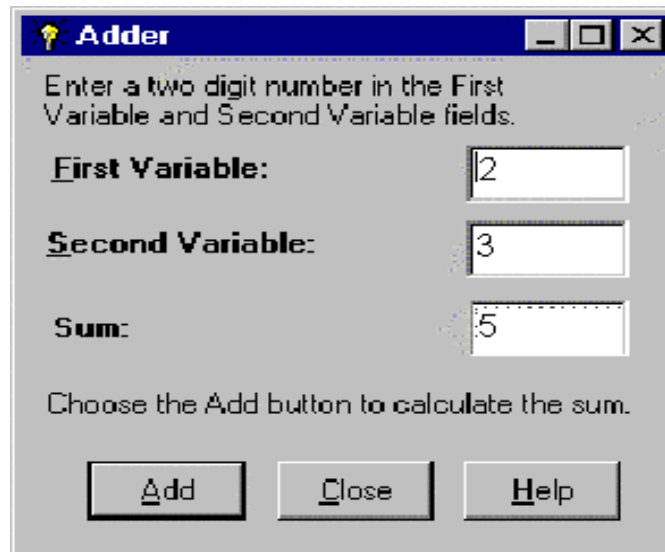
**First Variable:**

**Second Variable:**

**Sum:**

Choose the Add button to calculate the sum.

<i>What you do</i>	<i>What happens</i>
Open <b>ADDER</b>	The Adder application opens.
Press <b>2</b>	A <b>2</b> appears in the field
Press <b>Enter</b>	The focus is in the second field.
Press <b>3</b>	<b>3</b> appears in the second field.
Press <b>Enter</b>	A <b>5</b> appears in the third field.



The screenshot shows a window titled "Adder" with a yellow key icon. The window contains the following text and controls:

Enter a two digit number in the First Variable and Second Variable fields.

**First Variable:**

**Second Variable:**

**Sum:**

Choose the Add button to calculate the sum.

At the bottom, there are three buttons: "Add", "Close", and "Help".

# Equivalence Class & Boundary Analysis

There are  $199 \times 199 = 39,601$  test cases for valid values:

- definitely valid: 0 to 99
- might be valid: -99 to -1

There are infinitely many invalid cases:

- 100 and above
- -100 and below
- anything non-numeric

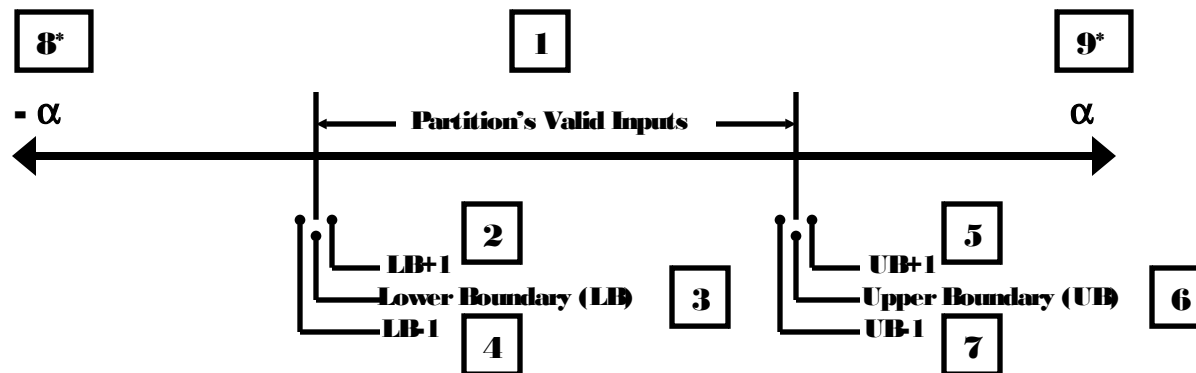
» Testing Computer Software, pages 4-5 & 125-132

# **Equivalence Class & Boundary Analysis**

- **Equivalent Class Partitioning and Boundary Condition Analysis**
  - **Equivalence Class**—Two tests belong to the same equivalence class if the expected result of each is the same. Executing multiple tests of the same equivalence class is by definition, redundant testing.
  - **Boundaries**—Mark the point or zone of transition from one equivalence class to another. The program is more susceptible to errors at the boundary conditions. Therefore, these are powerful sets of tests within the equivalent class to use.
  - Generally, each class is partitioned by the boundary values. Nevertheless, not all equivalence classes have boundaries. For example, the browser equivalence classes consist of Netscape Navigator class and Microsoft Internet Explorer class.

*Read Testing Applications on the Web, 2<sup>nd</sup> Edition, Wiley for more information*

\* Smallest/Largest Possible Values Allowed via UI



- For each equivalence class partition, we'll have at most, 9 tests to execute.
- It is essential to understand that each identified equivalence class represents a specific risk that it may pose.

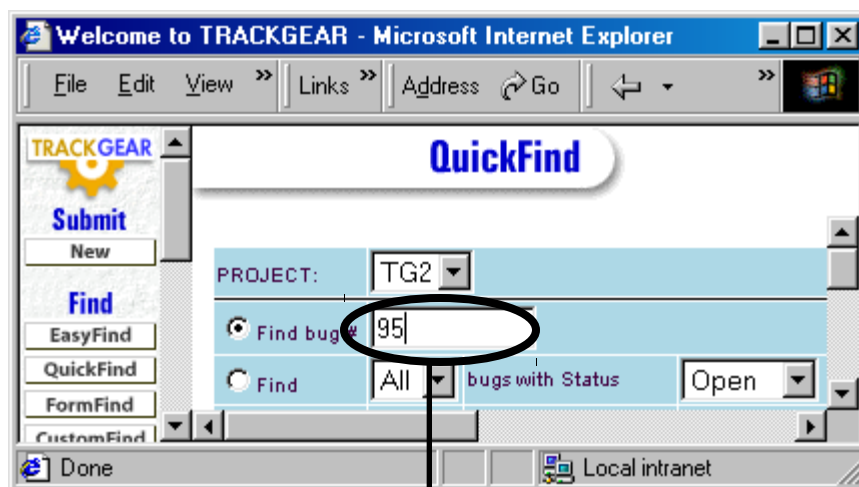
## **General Steps**

- Identify the classes
- Identify the boundaries
- Identify the expected output(s) for valid input(s)
- Identify the expected error-handling for invalid inputs
- Generate a table of tests (maximum, 9 tests for each partition of each class).



# Test Design

## Example 1:



**Valid Values: 1 to 9999  
or 1 to 4-digit value**

Test Case	Input	Output
<b>Value Class Partition</b>		
1	Any Valid Input	Functional Result
2	2	Functional Result
3	1	Functional Result
4	0	Error Handling
5	10000	Error Handling
6	9999	Functional Result
7	9998	Functional Result
8	-99999...	Error Handling
9	99999...	Error Handling
<b>Number of Character Class Partition</b>		
1	Any Valid Input	Functional Result
2	2	Functional Result
3	1	Functional Result
4	NULL	Error Handling
5	5	Error Handling
6	4	Functional Result
7	3	Functional Result
8	N/A	N/A
9	99999...	Error Handling

# Equivalent Class Partitioning

## Example 2:

A programmer implements the range 1 to 12 at an input, which stands for the month January to December in a date, has in his code a line checking for this range. This may look like:

```
if (month > 0 && month < 13)
```

But a common programming error may check a wrong range, starting the range at 0 by writing:

```
if (month >= 0 && month < 13)
```

For more complex range checks in a program this may be a problem which is not so easily spotted as in the above simple example.

Applying boundary value analysis

To set up boundary value analysis test cases you first have to determine which boundaries you have at the interface of a software component. This has to be done by applying the equivalence class technique. Boundary value analysis and equivalence partitioning are inevitably linked together. For the example of the month in a date you would have the following partitions:

...	-2	-1	0	1	.....	12	13	14	15	.....
----- ----- -----										
invalid partition 1			valid partition				invalid partition 2			

# Objectives of Testing

- Your objective **should not** be to verify that the program works correctly because:
  - If you can't test the program completely, you can't verify that it works correctly.
  - The program *doesn't* work correctly, so you can't verify that it does.
  - Then as a tester, you fail to reach your goal every time you find an error.
  - You'll be more likely to miss problems than if you want and expect the program to fail.

# Objectives of Testing

The Objective of Testing a Program is to Find Problems.

Finding problems is the core of your work. You should want to find as many problems as possible. The more serious the problem tester find, the better tester is.

***A test that reveals a problem is a success. A test that did not reveal a problem is (often) a waste of time!***

# Objectives of Testing

The Purpose of Finding Problems is to Get Them Fixed.

The point of the exercise is quality improvement!

- ***The best tester is not the one who finds the most bugs or who embarrasses the most programmers.***
- ***The best tester is the one who gets the most bugs fixed.***

- We cannot test everything, thus,
  - We want to focus and use our testing time in finding as many bugs as possible through good test design
  - There is no such thing as bug-free software
- We want to cover as many areas of the application as possible within the time constraint.
- We want to find the most serious bugs as quickly as possible.
- We want to be very familiar the test design technique called equivalent class and boundary condition analysis. It is an effective way to find bugs quickly.

# An Overview of Testing

2.1 Testing Group

2.2 Objectives of Testing

**2.3 Test Planning Framework**

2.4 Test Coverage

2.5 Manual Testing vs. Automated Testing

2.6 Key Lessons-Learned

# **Testing Paradigm Definitions**

## **Glass-box Testing**

In glass box testing, the tester (usually the programmer) uses his/her knowledge of the source code to create test cases.

- Hung Nguyen

Tests design based on the knowledge of the code to exercising the code

- Doug Hoffman

Glass-box Test Case example:

- Testing the piece of code for the Yahoo login functionality.
- This test type is usually executed by the developer.



## CHAPTER 2.3

Black box tests execute the running program, without reference to the underlying code. This is testing from the customer's view rather than from the programmer's.

- Hung Nguyen

Focus on input, outputs, and an “externally derived” theory of operation

- Cem Kaner

Black-box Test Case example:

Testing the Yahoo Mail login:

Step	Description	Data	Expected Result
1	Open IE and go to http://mail.yahoo.com		Yahoo page is displayed.
2	Enter a valid yahoo ID into the 'Yahoo ID' text box	Yahoo ID: kimlgr1	
3	Enter a valid password into the 'Password' text box	Password: logigear	
4	Click on the 'Sign In' button		Login is successful.



The screenshot shows the Yahoo! Mail login interface. At the top right, there is a blue banner that says "Prevent Password Theft". The main heading is "Sign in to Yahoo!". Below this, there are two text input fields: "Yahoo! ID:" and "Password:". Under the password field, there is a checkbox labeled "Remember my ID on this computer". A "Sign In" button is located below the checkbox. Below the button, there is a link that says "Why this is secure". At the bottom of the login section, there is a link that says "Forget your ID or password? | Help". Below the login section, there is a section titled "Don't have a Yahoo! ID?" with the text "Signing up is easy." and a "Sign Up" link.

## CHAPTER 2.3

Gray-box testing consists of methods and tools derived from the knowledge of the application internals and the environment with which it interacts, that can be applied in black-box testing to enhance testing productivity, bug finding and bug analyzing efficiency.

- Hung Nguyen

Testing involving inputs, outputs, but test design is educated by information about the code and the program operation of a kind that would normally be out of scope of the view of the tester.

- Cem Kaner

Gray-box Test Case example:

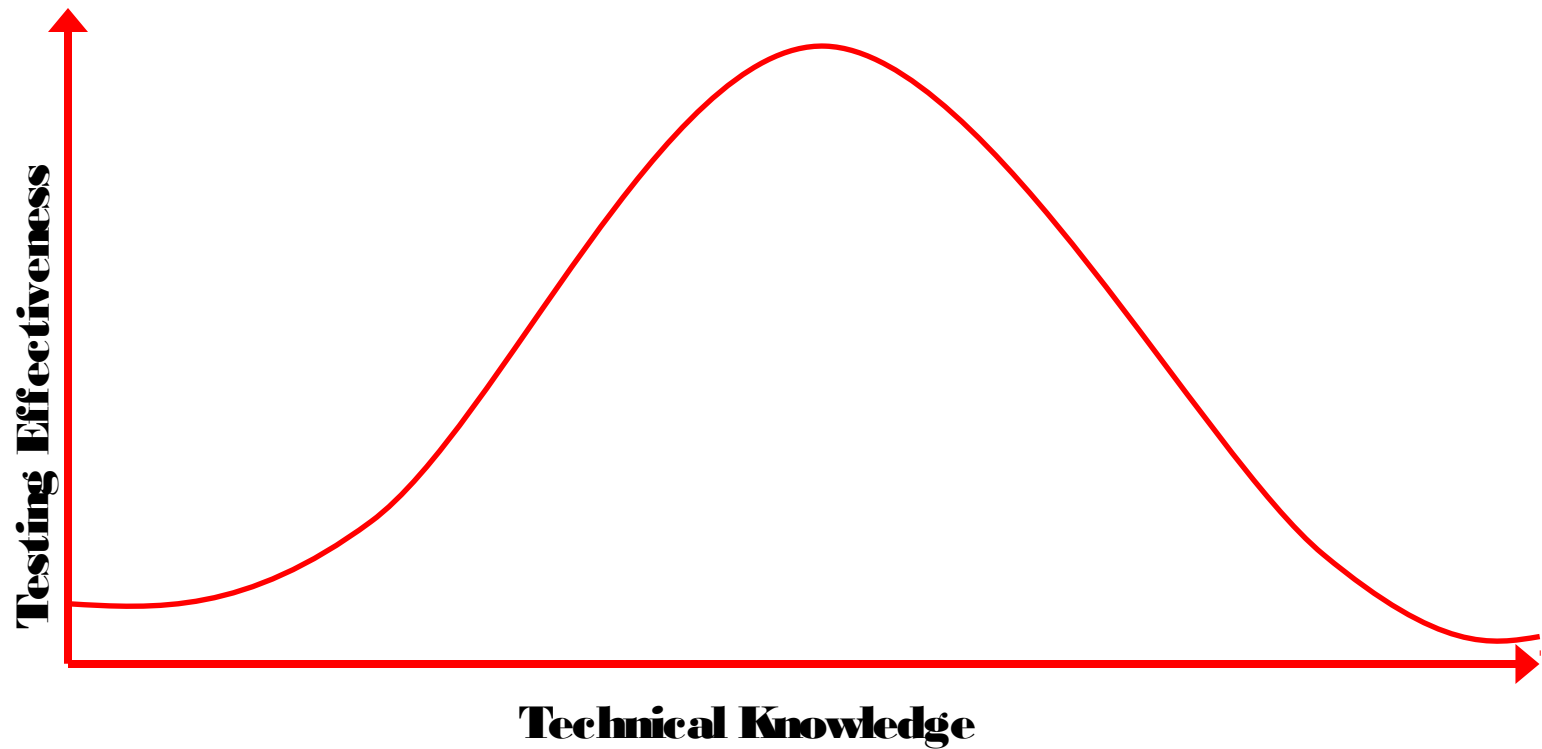
Testing the Yahoo login with HTML commands

Step	Description	Data	Expected Result
1	Open IE and go to http://mail.yahoo.com		Yahoo page is displayed.
2	Enter an HTML command into the 'Yahoo ID' text box	Yahoo ID: </BODY>	
3	Enter another HTML command into the 'Password' text box	Password: </HTML>	
4	Click on the 'Sign In' button		User-friendly error messages displays to inform about the invalid yahoo ID/ password.

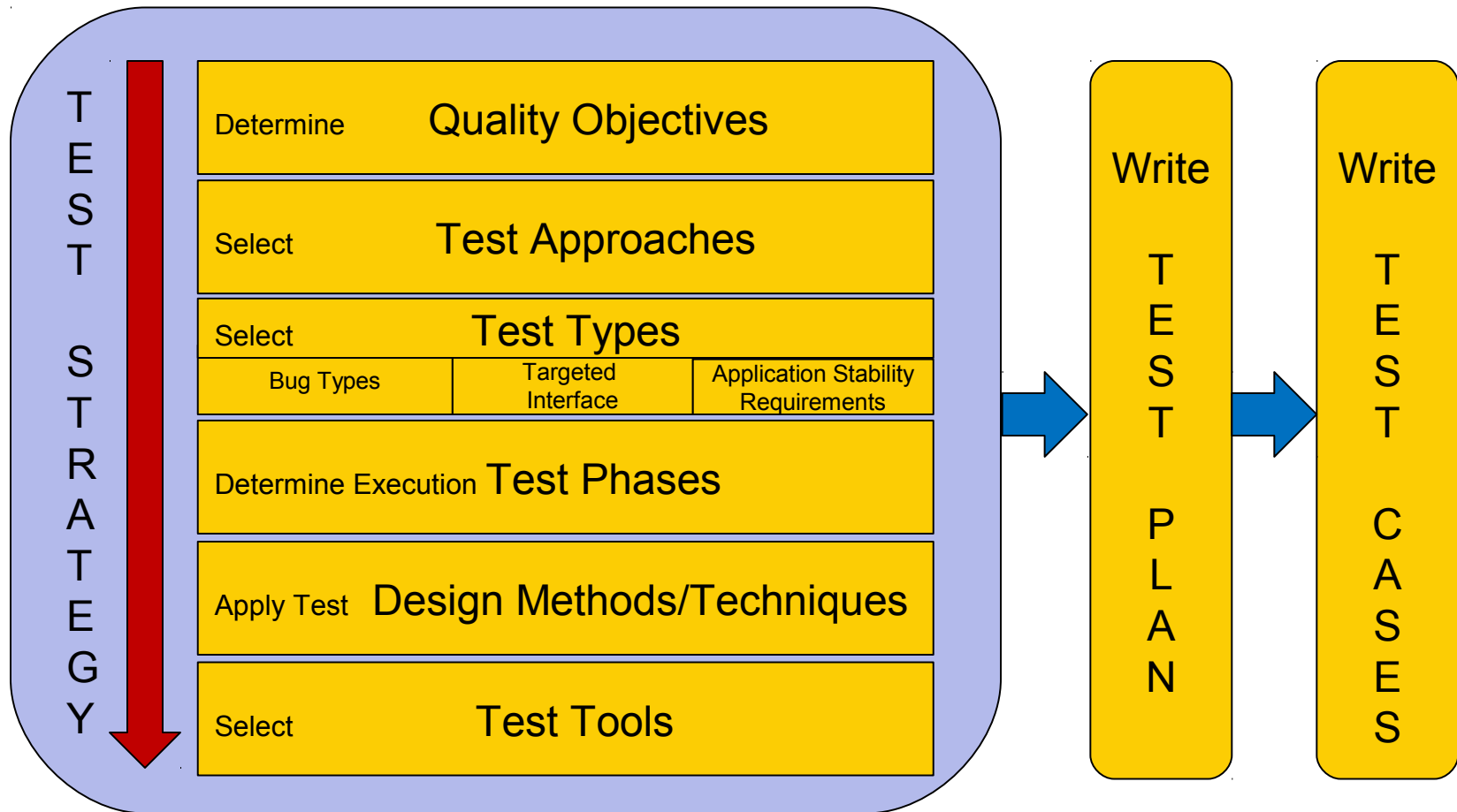


The screenshot shows the Yahoo! Sign In page. At the top right, there is a blue banner that says "Prevent Password Theft". The main heading is "Sign in to Yahoo!". Below this, there are two input fields: "Yahoo! ID:" and "Password:". Below the "Password:" field, there is a checkbox labeled "Remember my ID on this computer". To the right of the checkbox is a "Sign In" button. Below the "Sign In" button, there is a link that says "Why this is secure". Below that, there is a link that says "Forget your ID or password? | Help". At the bottom, there is a section titled "Don't have a Yahoo! ID?" with the text "Signing up is easy." and a "Sign Up" link.

# Testing Effectiveness



# Test Planning Framework Diagram



## CHAPTER 2.3

- **Definition:** A holistic plan that starts with a clear understanding of the core objective of testing from which, we derive a structure for testing by selecting from many testing styles and approaches available to help us meet our objectives. There are three focal points (SP<sup>3</sup>) that one can make adjustment to—People, process, and practice (methods and tools) to get the best results. “Process” means the ways of doing things that people have agreed to. “Practice” means a successful combination of testing methods and tools that help meet the agreed objectives. “People” means the human resource that is capable of applying the practice. The description of key factors which are important during the preparation and execution of the test.
- Test Goal – Why are we testing?
  - Minimize number of defects released to production?
  - Validate Requirement?
  - Find Bugs to improve quality?
  - Find Design issues?
  - Maximize customer satisfaction, by giving the customers the features they want?
  - Other?

# **Test Strategy**

- Goals for testing usually fall into 2 categories: Bug finding or validation. Once you have settled upon the goal for the test project, you can strategize how you will accomplish that goal.
- The test strategy describes how you will test the application.

# Quality Objectives

**Definition:** Quality objective is concrete requirement term that specifies the type of the quality objective and the level of acceptability for each.

**Example:** Usability, security, environment, communication, stress, volume, functionality, operations, recovery and performance

# Test Approaches

**Definition:** A testing philosophy or style that drives the selection of certain test design methods.

**Examples:**

Requirement-based, Exploratory, Scenario-based, Model-based, Attack-based, Risk-based, Fault Injection, DAST, etc.



**Definition:** A category of test activities with the objective of exposing specific types of bugs at certain interface, applicable for various level of product maturity. The selection of the test types must satisfy the test or quality objectives.

**Examples:** Requirement Review, Design Review, Code Review, Code Walkthrough, Design Walkthrough, Unit, Module, API, External Functional, Usability, Accessibility, Configuration, Compatibility, I18N, L10N, Regression, Performance, Load, Stress, Soak, Failover/Recovery, Installation, Security, Data verification testing, Compliance, etc.

(...will be discussed more in later session)

**Definition:** Bugs or software errors are often grouped into categories or types so we can condense a huge list of issues into a conceptually manageable group of ideas. Any approach to categorizing bugs is arbitrary.

**Examples:** User Interface, Error Handling, Boundary-Related, Calculation, Initial and Later States, Control, Flow, Handling or Interpreting, Data, Race Conditions, Load, Conditions, Hardware/Environment Compatibility, Source, Version, and ID Control ,Testing, Documentation  
(...will be discussed more in later session)

# Targeted Interface

**Definition:** A boundary across which, two independent systems meet and act on or communicate with each other. In computer technology, there are several types of interfaces. ([www.webopedia.com](http://www.webopedia.com))

**Examples:** Source-level (static testing such as code review and analysis), Calling Functions and Classes, API, GUI, CLI

# Application Stability Requirements

**Definition:** The state or quality of being stable of a software application. In software development, a stable application or functionality in an application is one that is reliable and dependable, and will consistently produce the same expected outputs given on the same set of inputs and conditions.

**Example:** Software application which has passed the alpha testing phase is more stable than when it was in the beginning of the alpha testing phase.

# Test Phases

**Definition:** A test phase is a time block often derived from the SDLC phases. Within each SDLC phase, there might be more than one test phase. Each test phase has different objectives related to bug finding or validation, described in each section. Testing staff participates in ongoing project management activities.

**Examples:** Pre-coding, Coding, Pre-integration, System, System, Integration, Final, Beta/User Acceptance, Post, Production

# Test Design Methods/Techniques

**Definition:** A test design method is a systematic procedure in which you create tests to be executed. A test type or test approach may use one or more test design methods.

**Examples:**

- Boundary condition and equivalent class analysis
  - Decision-tree/table (variables in combination), Combinatorial design
  - State-transitioning
- (...will be discussed more in later session)

## **Examples:**

- TestArchitec® for Action-Based Test (keyword test automation for functionality and regression testing)
- Webload® for load and performance testing
- InCtrl® for environment change analysis

- **Definition:** A management document outlining risks, priorities, and schedules for testing. A document prescribing the approach to be taken for intended testing activities.



## CHAPTER 2.3

- **Definition:** A specific of test data and associated procedures developed for a particular objective, such as to exercise a particular program path or to verify compliance with a specific requirement (IEEE – 729-1983)
- **Definition:** A test that (ideally) executes a single well defined test objective (i.e., a specific behavior of a feature under a specific condition) (Testing Computer Software – Kaner, Falk, Nguyen)

...will be discussed more in a later session.

## CHAPTER 2.3

- There are 3 common paradigms in testing engineering: glass-box testing, black-box testing, white-box testing.
  - In glass box testing, the tester (usually the programmer) uses his/her knowledge of the source code to create test cases
  - Black box tests execute the running program, without reference to the underlying code. This is testing from the customer's view rather than from the programmer's
  - Gray-box testing consists of methods and tools derived from the knowledge of the application internals and the environment with which it interacts, that can be applied in black-box testing to enhance testing productivity, bug finding and bug analyzing efficiency.
- LogiGear uses a Test Planning Framework to build testing strategy. In which, we determine Quality Objectives, select Test Approaches, Test Types, Bug Type, Targeted Interface, Application Stability Requirements, determine Execution Test Phases, apply Test Design Methods/Techniques, select Test Tools....to write test plan and test cases

# An Overview of Testing

2.1 Testing Group

2.2 Objectives of Testing

2.3 Test Planning Framework

**2.4 Test Coverage**

2.5 Manual Testing vs. Automated Testing

2.6 Key Lessons-Learned

# Test Coverage

We know:

Complete Testing is impossible.

There are too many tests to run.

We have limited time.

So, what do we test? How deep? Where? With what data?  
On what platforms? ...

# Coverage Review

**Coverage** measures of the amount of testing done of a certain type. Since testing is done to find bugs, coverage is a measure of your effort to detect a certain class of potential errors.

***So what kind(s) and level(s) of coverage are considered appropriate? There is no magic answer.***

# More on Coverage

Basic or simplistic coverage is the percentage of product “tested” based on some aspect of the project.

$$\text{Coverage} = \frac{\text{\# of tests to execute}}{\text{total \# of tests}}$$

# Test Coverage

Let's look at possible coverage metrics, or measurements of how much is tested and how much is not.

Quantity of testing	External	functionality	requirements
			functionality
			screens
			windows
			use cases
			forms
			user scenarios
			...
	Internal	test cases	
			data
			platforms
		modules	
		lines of code	
		number of APIs	
		# of parameters of each API	
		...	
		test cases	
			data
			platforms

**An analysis method that determines which parts of the software have been executed (covered) by the test case or suite and which parts have not been executed and therefore, may require additional attention.**

Various flavor of coverage:

**Function coverage** - Has each function in the program been executed?

**Statement coverage** - Has each line of the source code been executed?

**Condition coverage** - Has each evaluation point (such as a true/false decision) been executed?

**Path coverage** - Has every possible route through a given part of the code been executed?

**Entry/exit coverage** - Has every possible call and return of the function been executed?



## CHAPTER 2.4

- IEEE Unit Testing Standard is
  - 100% Statement Coverage
  - 100% Branch Execution(IEEE Std. 982.1-1988, § 4.17, “Minimal Unit Test Case Determination”)
- Most companies do not achieve this.
- Several people seem to believe that complete statement and branch coverage means complete testing. Or at least, sufficient testing. It is a useful measurement but not a perfect measure of *complete coverage*.

# Problems of Coverage

A common metric requested from test teams today is test coverage. There are often problems with this metric.

- If you have requirements traceability (mapping) or dynamic code coverage measurements from test case execution, test case coverage has meaning. Other than that, test case coverage can give a false sense of security.
- What are the test cases tied or linked to, if not requirements or code? User Scenarios? Paths (workflows)? What? Were the test cases approved as being adequate to cover the development for this release?

# An Overview of Testing

2.1 Testing Group

2.2 Objectives of Testing

2.3 Test Planning Framework

2.4 Test Coverage

**2.5 Manual Testing vs. Automated Testing**

2.6 Key Lessons-Learned

# Manual Testing & Automated Testing

- **Manual Testing** is performed by carrying out all the actions manually against the application under test (AUT), step by step and revealing whether a particular step was completed successfully or failing.
- Manual testing is always a part of any testing effort. It is especially useful in the initial phase of software development, when the software and its user interface are not stable enough, thus test automation does not make sense.



# Manual Testing & Automated Testing

## Automated Testing

- Using of software to control the execution of tests, and the comparison of actual outcomes with the expected (specified) outcomes.
- Setting up test-preconditions, and other test control and test reporting functions.
- Commonly, test automation involves automating a manual process already in place that uses a formalized testing process.  
([http://en.wikipedia.org/wiki/Test\\_automation](http://en.wikipedia.org/wiki/Test_automation))
- Software testing assisted by software tools that enable test engineers to pre-prescribe test cases: (1) the input data and the interfaces of the UAT which will receive the data, (2) the expected/referenced output data value, (3) where to extract the output data from the UAT interfaces, and which output data will be compared against the referenced output data to determine passed/failed. At run-time, the software tools will read the pre-prescribed test cases, execute them, analyze and evaluate the results to determine passed/failed automatically. Common application of automated testing includes, but not limited to functional and/or regression testing. (LogiGear)



# An Overview of Testing

2.1 Testing Group

2.2 Objectives of Testing

2.3 Test Planning Framework

2.4 Test Coverage

2.5 Manual Testing vs. Automated Testing

**2.6 Key Lessons-Learned**

# **Key Lessons-Learned**

- The Program Doesn't Work. Your task is to find errors.
- Be Methodical
- Complete Testing is Impossible
- Use Powerful Representatives of Tests
- Communication Must be Effective
- Change is Normal

# The Program Doesn't Work

- All programs have bugs. Nobody would pay you to test if their program didn't have bugs.
- Any change to a program can cause new bugs, and any aspect of the program can be broken.
- You DON'T "verify that the program is working." You FIND bugs.

*If you set your mind to show that a program works correctly, you'll be more likely to miss problems than if you want and expect the program to fail.*



# Be Methodical

- Testing isn't just banging away at the keyboard.
- To have any hope of doing an efficient job, you must be methodical:
  - Break the testing project down into tasks and subtasks so that you have a good idea of **what has to be done and what has been done**.
  - Track what has been done so that people don't repeat the same tasks and you know what tasks are left.
  - Prioritize the tasks.

# Complete Testing is Impossible

- There are a nearly infinite number of paths through any non-trivial program.
- There are a virtually infinite set of combinations of data that you can feed the program.

**You can't test them all.**

- Therefore, your task is to find bugs -- not to find ***all*** the bugs.
- You want to
  - Find as **many** bugs as possible
  - Find the most **serious** bugs
  - Find bugs as **early** as possible

# Use Powerful Representatives of Tests

Develop powerful tests by analyzing

- Equivalence classes
- Boundary conditions
- Input combinations
- Data/functionality relationships
- ...

*(These will be discussed in later sections)*

# **Communication Must be Effective**

- Your bug reports advise people of difficult situations. The problems that you report can affect
  - The project schedule
  - The company's cash flow
- The clearer your reports are, the more likely it will be that the company will make reasonable business decisions based on them.
- Persuasive and technical writing, oral argument, face-to-face negotiation, and diplomacy are core skills for your job.

# Change is Normal

- Project requirements change, as do design and specifications.
- The market changes.
- Development groups come to understand the product more thoroughly as it is being built.
- Some companies accept late changes into their products as a matter of course.

**As a new tester, you might decide quickly that this is a bad, amateurish way to do business.**

**It might be, and it might not be.**

***Take steps to make yourself more effective in dealing with late changes.***

- Testing Group includes QA, QC, and Testing staff who perform very important services such as: find and report bugs, communicate problems, and evaluate the product quality...
- The objective of testing is finding as many good bugs as possible.
- For testing software, LogiGear uses a test planning framework. This framework includes several parts: test strategies, approaches, test plans, test methods, test cases...



## **CHAPTER 3**

# **Test Requirement**

- Product's Document
- What is Test Requirement (TR)?
- Test Requirement Essentials
- Requirement Attributes



# OBJECTIVES

- An introduction to requirement, specification and design
- An introduction to test requirements and how to use test requirement to implement test case

# Test Requirement

## **3.1 Product's Document**

3.2 What is Test Requirement (TR)?

3.3 Test Requirement Essentials

3.4 Requirement Attributes

# Product's Documents

## **Definitions:**

Requirements - the customer's problem or desires

Specification - what you intend to make to solve the problem or meet the desire

Design - how the thing works

## **Examples:**

Requirement: the room should be warm

Specification: The room will be kept between 25 and 28 degrees.

Design: Heat pump, augmented by electric heater...

# Test Requirement

3.1 Product's Document

**3.2 What is Test Requirement (TR)?**

3.3 Test Requirement Essentials

3.4 Requirement Attributes

# Test Requirement

- A statement of what should be tested in the AUT
- Functional Requirement: the requirement for the functions that the application should do
- Non-functional requirement: the requirement for the properties that the functions should have or should look like. There are 3 types of non-function TR:
  - Look n feel
  - Boundary
  - Negative

# Test Requirement

3.1 Product's Document

3.2 What is Test Requirement (TR)?

**3.3 Test Requirement Essentials**

3.4 Requirement Attributes

# Test Requirements Essentials

The formats can be very different. What they look like is unique to every company and writer. They are hopefully full of useful information.

- Target Platform
- System Requirements
- A description of the functionality
- Intended Use
- Expected Problem Areas

# Test Requirement

3.1 Product's Document

3.2 What is Test Requirement (TR)?

3.3 Test Requirement Essentials

**3.4 Requirement Attributes**



# Test Requirement Attributes

- Boehm (1984) lists the attributes of good software requirements
  - Written
  - Specific
  - Complete
  - Correct
  - Feasible
  - Consistent
  - Prioritized
  - Clear/ Unambiguous
  - **Verifiable (key attribute)**
  - Concise
  - In Understandable language

- Can save a file after opening a new file
- Can save a file with text (use your name) and text with space, such as michealhackett and micheal hackett
- Can save in the following format
  - XML
  - Web page
  - Plain text
  - Works 6

Application Under Test: Yahoo Mail

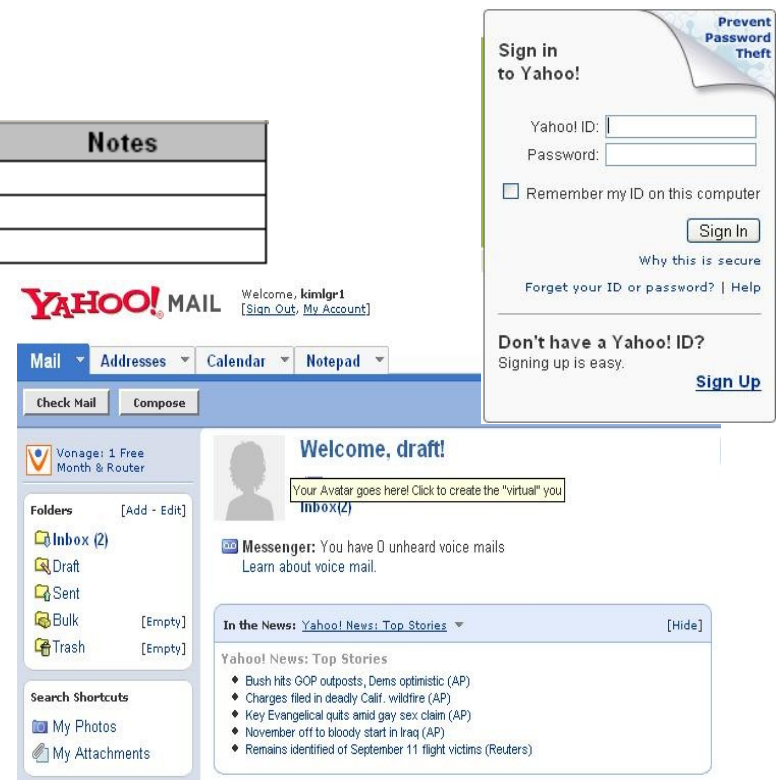
Let's practice create some Test Requirements for the Yahoo Mail.

Below is an example.

TR-ID	Test Requirements	Notes
TR001	User is able to login with valid username and password.	

### Exercise:

Please create 3 more Test Requirements for the Yahoo Mail.



- Documents of a project usually has 3 types: requirement, specification and design.
- A test requirement can be written from a user story or from other sources of the software. Tester uses test requirement to implement test cases.

(We will learn more about how to create a test requirements from user story in LCTP II)



## CHAPTER 4

# Test Methods and Test Case Design Techniques

## CHAPTER 4

- Some Common Test Approaches and Test Types
  - Requirement Testing
  - Exploratory Testing
  - Smoke Testing
  - Regression Testing
- Test Case
  - Test case criteria
  - Test case essentials
  - Test case syntax
- Some Test Case Design Techniques
  - Equivalent Partitioning
  - Boundary Analysis
  - Constraint Analysis
  - State Transition
  - Condition Combination

# OBJECTIVES

- An introduction to commonly used test methods, test types and how to apply them to test an application
- An overview of test case and some techniques to design test case

## **4.1 Some Common Test Methods and Test Types**

- Requirement Testing
- Exploratory Testing
- Smoke Testing
- Regression Testing

## **4.2 Test Case**

- Test case criteria
- Test case essentials
- Test case syntax

## **4.3 Some Test Case Design Techniques**

- Equivalent Partitioning
- Boundary Analysis
- Constraint Analysis
- State Transition
- Condition Combination



## CHAPTER 4.1

- Definitions: A definitive procedure for the identification, measurement, and evaluation of a material, product, system, or service that produces a test result.  
(ASTM – American Society for Testing and Materials)

# **Common Test Methods and Test Types**

- Test approaches:
  - Requirement-based Testing (Passive Testing)
  - Exploratory Testing (Ad Hoc Testing)
  - ....

The best practice is a combination of a few methods to satisfy the project needs
- Test Types
  - Regression Testing
  - Smoke Testing
  - Stress Testing
  - Load Testing
  - Performance Testing
  - ...

# **Requirement-based Testing**

- Requirement-based Testing is:
  - Testing the requirement
  - Testing the specifications
  - Testing the product based in the information in the requirement and spec documentation
- The primary goal of Requirement-based Testing is Requirement Validation. It is not experimental, behavioral, exploratory or a bug-finding exercise in nature. It validates.

# **Requirement-based Testing**

## **The “Mechanical Approach” for Test development:**

- Analyze requirements
- Gather more information
- Make at least a test case for every requirement
- Execute the test cases

# Exploratory Testing

- Exploratory Testing can have many names: Ad hoc, Discovery, Error Guessing...
- The primary goal of exploratory testing is to uncover new defects in the product on-the-fly.
- The task is finding ways to creatively design experiments, exploring, experimenting, going from a known place to an unknown place.

# Exploratory Testing

- Exploratory Testing is not requirement-driven but it is intense testing. The goal is to **probe** for **weak areas** of the program. Some various error conditions should be checked are:
  - Out of boundary
  - Null input and overflow (storage of the sum)
  - Non-numeric
  - Hardware and memory error handling
  - ...
- Exploratory Testing is the testing approach which can produce the highest number of bugs

# Regression Testing

- Most common type of work we do!
- Re-run
- Quality Control- not testing, not *bug finding*
- Expensive
- First to automate

# **Regression Testing**

- Regression Testing: The testing that is performed after making a functional improvement or repair to the program.
- Regression Purpose:
  - Selective re-run the test cases to verify that the modifications have not caused any unintended effects and that system still complies with its specified requirements.
  - Re-run the test cases to make sure that the bugs marked as fixed by developers are really fixed.
- Re-test all test cases can be done using automated testing tools.



# Regression Testing

- Some of the most popular strategies for selecting regression test suites:
  - Retest all: Rerun all test cases. Simple but impossible in the time that we have in our everyday practice.
  - Retest Risky Use Cases: Choose baseline tests to re-run by risk heuristics.
  - Retest by Profile: Choose baseline tests to re-run by allocating time in proportion to operational profile.
  - Retest Changed Segment: Choose baseline tests to re-run by comparing code changes.

# Smoke Testing

- A *smoke test* is a collection of written tests that are performed on a system prior to being accepted for further testing, also known as *build verification test*.
- This is a "shallow and wide" approach to the application. The tester "touches" all areas of the application without getting too deep, looking for answers to basic questions like, "Can I launch the test item at all?", "Does it open to a window?", "Do the buttons on the window do things?".
- There is no need to get down to field validation or business flows. If you get a "No" answer to basic questions like these, then the application is so badly broken, there's effectively nothing there to allow further testing.
- These written tests can either be performed manually or using an automated tool.

# **Common Types of Black Box Testing**

- Acceptance (into-testing) Test
- Feature-Level Test
- Regression Test
- Configuration & Compatibility Test
- Documentation and Online Help Test
- Utilities/ Tool Kits and Collateral Test
- Install/ Uninstall Test
- Integrity & Release Testing
- Import/ Export
- User Interface Test
- Usability test
- Performance Test
- Benchmark Test
- Acceptance by Customer
- Beta Testing
- Initial Stability Assessment

- Different test methods have different goals:
  - Projects with lots of documentation tend to use requirements based testing as the test development methods.
  - Projects with little documentation rely on ad hoc/exploratory testing.
  - Projects in later phases of release (iteration, functional increments) rely more on regression testing.

## 4.1 Some Common Test Methods and Test Types

- Requirement Testing
- Exploratory Testing
- Smoke Testing
- Regression Testing

## 4.2 Test Case

- **Test case criteria**
- **Test case essentials**
- **Test case syntax**

## 4.3 Some Test Case Design Techniques

- Equivalent Partitioning
- Boundary Analysis
- Constraint Analysis
- State Transition
- Condition Combination

- **Definition:** A test that (ideally) executes a single well defined test objective (i.e., a specific behavior of a feature under a specific condition) (Testing Computer Software – Kaner, Faulk, Nguyen)
- **Definition:** A specific set of test data and associated procedures developed for a particular objective, such as to exercise a particular program path or to verify compliance with a specific requirement (IEEE 729-1983)

# Why write test case?

- Accountability
- Reproducibility
- Tracking
- Automation
- To find bugs
- To verify that tests are being executed correctly
- Use as a Training Tool for new testers
- For compliance
- To measure test coverage

# Test Case Criteria

- An excellent test case satisfies the following criteria:
  - Reasonable probability of catching an error
  - Neither too simple nor too complex
  - Not redundant with other tests (think of equivalence classes – see later)
  - Best of its breed (think of boundary conditions – see later)
  - Makes program failures obvious



# **Good and Not-so-Good Tests**

## **A Good Test:**

- Refers and maintains tight control over specific test data
- Has detailed enough Test Design Steps so that any testers with basic knowledge of the system can execute the test
- Is aware of the Tester's experience
- Has clear criteria for Pass or Fail

## **A Not-so-Good Test**

- Leaves it up to the user to find test data
- Gives very high level instructions that leave too much room for "artistic interpretation"
- Does not consider the Tester's experience
- Leaves out follow-up verification steps which make it difficult to determine Pass or Fail criteria.

## CHAPTER 4.2

### Things usually to include in a test case:

- Tracking Information
- Test Case ID
- Test Description
  - Purpose/Objective/Title
  - Procedures/Steps
  - Script
- Parameters
  - Input
  - Output
  - Default
  - Options
  - Conditions
- Call to system (printer, system clock, available RAM, APIs)
- Expected Result
- Observed Result
- Pass/Fail/Blocked/Skipped
- Bug ID
- Notes/Comments
- Environment

# **Test Case Objective**

- The most important part of a test case is the 1-line title describing the objective of the test
- That 1-line title can be called:
  - Test Title
  - Test Name
  - Test Case
  - Test Objective
  - Test Goal/Purpose

# **Test Case Objective**

- It is most important because
  - It gives the reader a description and idea of the test
  - A good test name makes review easier
  - Easier to pass to another person
  - Easier to pass to automation team
  - Describes intention of the test
  - In many cases, may be the only part of a test case documented

# Test Case Objective Syntax

- Test Objective suggested syntax:  
**Action + Function + Operating Condition**
- In which:
  - Function may be function, feature, validation point
  - Condition may be data
  - Action:
    - Verify
    - Test
    - Validate
    - Prove
    - Execute
    - Print
    - Calculate
    - Run
    - ...any action verb

# Test Case Examples

Action	Function	Operating Condition
Run	annual report	from standard data (file location)
Run	annual report	on Day 1 of fiscal year
Run	annual report	from empty spreadsheet
Run	annual report	on last day of fiscal year

# Test Case Planning Template

TC ID	Description	Steps	Expected result	Observed result	Status
TC001	<i>/*Describe the test objective here.*/</i>	<i>/*Be very clear and specific step so that the test can be reproducible*/</i>  <u>Precondition:</u>  <i>/* describe the step to set a program with a specific ... */</i>  <u>Steps:</u> 1-Action 1  2-Action2	<i>/*The expected results must a be written in a very specific way</i>  You need to pre-determine what your program is supposed to do ahead of time*/	<i>/*write down the result that you get when execute test cases*/</i>	<i>/*Fill the status of this test Pass\Failed by compare the Expected result and Actual result8?</i>

# Validation Points

Tests need validation points.

- This is the expected result. It can also be stated in the test objectives.
- Do not rely on “seeing” that a test passes or fails. Write it.
- Many times it is easier to define the test once you clearly state what behavior, result or point you are attempting to validate.



# Test Methods and Test Case Design Techniques

## 4.1 Some Common Test Methods and Test Types

- Requirement Testing
- Exploratory Testing
- Smoke Testing
- Regression Testing

## 4.2 Test Case

- Test case criteria
- Test case essentials
- Test case syntax

## 4.3 Some Test Case Design Techniques

- **Equivalent Partitioning**
- **Boundary Analysis**
- Constraint Analysis
- State Transition
- Condition Combination

# **Some Test Case Design Techniques**

- Equivalence Partitioning
- Boundary Value Analysis
- State Transition/ Model-Based Testing
- Cause-Effect Grapping
- Syntax Testing
- Statement Testing
- Branch/Decision Testing
- Data Flow Testing
- Branch Condition Testing
- Branch Condition Combination Testing
- Modified Condition Decision Testing
- LCSAJ Testing (Linear Code Sequence and Jump)
- Random (Ad hoc) Testing

# **Equivalence Class and Boundary Analysis**

- **Equivalence Class:** Two tests belong to the same equivalence class if the expected result of each is the same. Executing multiple test cases of the same equivalence class is by definition, redundant testing.
- **Boundaries:** Mark the point or zone of transition from one equivalence class to another. The program is more susceptible to errors at the boundary conditions. Therefore, these are powerful sets of test cases within the equivalent class to use.
- Generally, each class is partitioned by the boundary values. However, not all equivalence classes have boundaries. For example, the browser equivalence classes consist of Netscape Navigator class and Microsoft Internet Explorer class.

*It is important to recognize that two tests are equivalent only with respect to a specific risk.*

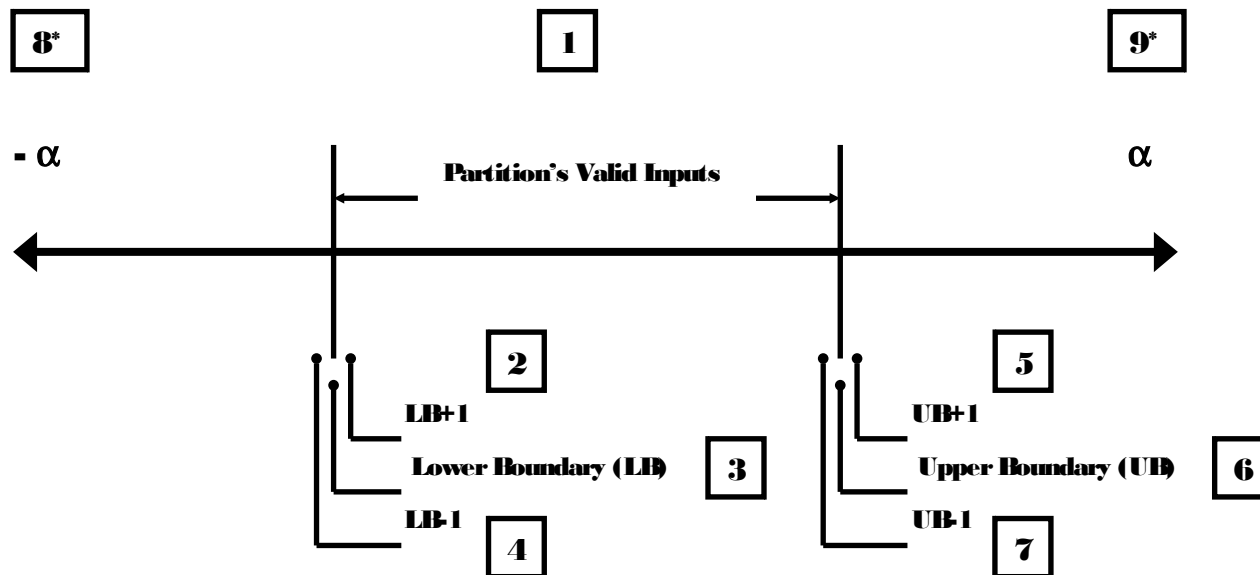
# Equivalence Class Example

- Two tests are equivalent if you expect the same result from each. Here are some examples:
  - Ranges of numbers (such as all the numbers between 10 and 99)
  - Membership in groups (dates, time, country names, etc.)
  - Invalid inputs
  - Equivalent output events (variation of inputs that produces the same outputs)
  - Equivalent operating environments
  - The number of times you've done something
  - The number of records in the database (or how many other equivalent objects)
  - Equivalent sums or other arithmetic results
  - Equivalent number of items entered (such as the number of characters entered into a field)
  - Equivalent space (on a page or on a screen) required
  - Equivalent amounts of memory, disk space, or other resources available to the program

# Equivalence Class and Boundary Analysis

- For each equivalence class partition, we'll have at most, 9 test cases to execute.
- It is essential to understand that each identified equivalence class represents a specific risk that it may pose.

\* **Smallest/Largest Possible Values Allowed via UI**



# **Equivalence Class and Boundary Analysis**

- **General Steps**
  - Identify the classes
  - Identify the boundaries
  - Identify the expected output(s) for valid input(s)
  - Identify the expected error-handling (EH) for invalid inputs
  - Generate a table of test cases (maximum, 9 test cases for each partition of each class).

# **Equivalence Class and Boundary Analysis**

- Other Equivalence Classes and Special Cases
  - Negative values
  - Number of digit or characters
  - 0
  - Non-printable characters
  - Upper ASCII (128-254)
  - O/S reserved characters
  - Space
  - Nothing
  - etc.
- See Input Dialog Test Matrix

# Character Groupings

## CHAPTER 4.3

<b>Low ASCII</b>	These are the Ctrl keys. Interesting ones include 000-null, 007-beep, 008-BS, 009-Tab, 010-LF, 011-VT/home, 012-FF, 013CR, 026-EOF (end of file, very nasty sometimes), 027-ESC	
<b>Non-alphanumeric, standard, printing ASCII characters. We often lump these together even though they're in four distinct groups.</b>	Low non-alphanumeric (ASCII codes 32-47)	space ! " # \$ % & ' ( ) * + , - . /
	Intermediate non-alphanumeric (ASCII 58-64)	: ; < = > ? @
	More intermediates (ASCII 91-96)	[ \ ] ^ _ `
	Top of standard ASCII (ASCII 123-126)	{   } ~
<b>Digits</b>	(ASCII 48-57)	0 1 2 3 4 5 6 7 8 9
<b>Upper and lower case alpha</b>	(ASCII 65-90)	A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
	(ASCII 97-122)	a b c d e f g h i j k l m n o p q r s t u v w x y z
<b>Upper ASCII</b>	(ASCII 128-254)	These subdivide further depending on the character set, the user's language, and the application.
<b>Modifier keys</b>	These keys include (depending on the keyboard) Alt, Right-Alt, Shift, Control, Command, Option, Left-Amiga, Right-Amiga, Open-Apple, etc. They generally have no effect when pressed alone, but when pressed in conjunction with some other key, they create a	
	It often pays to test all the "interesting" standard values, plus a sample of others.	
	It is often best to assign a separate chart column to each modifier key, i.e. one column for Ctrl, one for Shift, etc.	
<b>Function keys</b>	Test them alone and in combination with the modifier keys.	
<b>Cursor keys</b>	Test them alone and in combination with the modifier keys. It's common for every modifier key to have a different effect on cursor keys.	
<b>Numeric keypad keys</b>	These are not necessarily equivalent to number keys elsewhere on the keyboard.	
<b>European keyboards</b>	The left and right Alt keys often have different effects on non-English keyboards. Also, these keyboards provide dead keys – you press a dead key to specify an accent, then press a letter key and (if it's a valid character) the computer displays that let	



# Test Methods and Test Case Design Techniques

## 4.1 Some Common Test Methods and Test Types

- Requirement Testing
- Exploratory Testing
- Smoke Testing
- Regression Testing

## 4.2 Test Case

- Test case criteria
- Test case essentials
- Test case syntax

## 4.3 Some Test Case Design Techniques

- Equivalent Partitioning
- Boundary Analysis
- **Constraint Analysis**
- State Transition
- Condition Combination

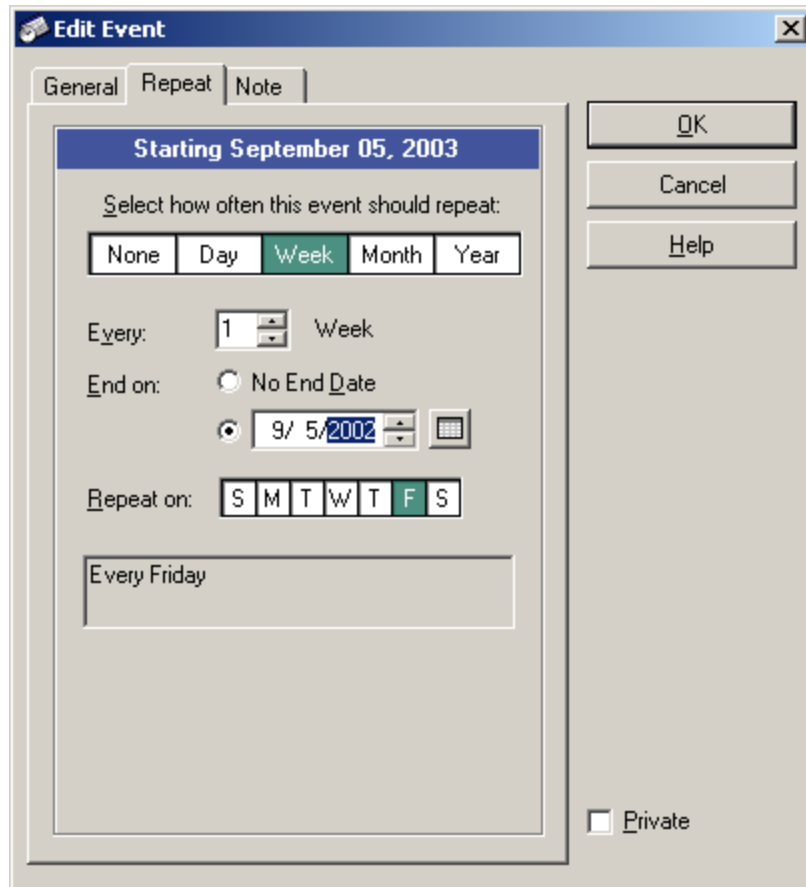
### Constraint Analysis

- Equivalence class and boundary analysis look at each variable (field) in isolation.
- This is important, but we also have to look at the relationship between different variables.
- Cause-Effect Graphing is one approach to this, but it can be complicated.
- Here is a simpler approach that is still quite valuable.

### General Steps

- Identify the dependent variables and study their respective constraints.
- Identify all possible input and out put interfaces for the variables
- List the variables in a tabular format with their input/output interfaces and their respective constraints.

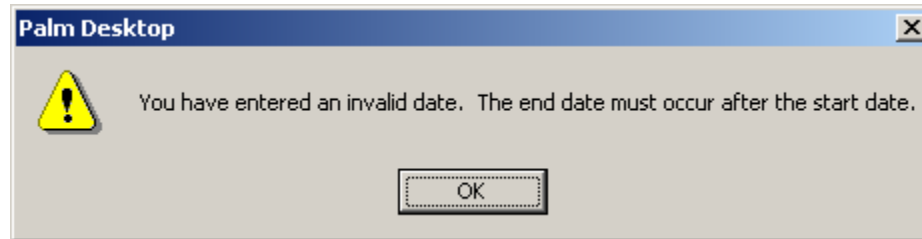
# Data Relationships



Look at Edit Event dialog box from the Palm Desktop Software.

Note that the End Date for repeating this event is *before* the Start Date. That is not possible.

# Data Relationships



The program checks the End Date against the Start Date and rejects this pair as impossible because the task can't end before it starts.

The value of End Date is *constrained by* Start Date, because End Date can't be earlier than Start Date.

The value of Start Date *constrains* End Date, because End Date can't be earlier than Start Date.

# Data Relationships

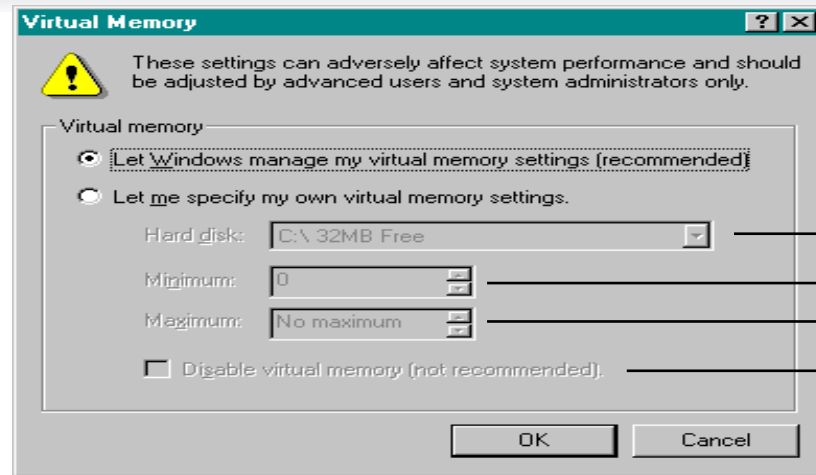
	INPUT	OUTPUT		RESTRICTIONS	
Data Field	Entry Source	Display	Print	Constrained by	Constraint
Start Date					End Date
End Date				Start Date	

This table shows how the variables are related. Here are the columns:

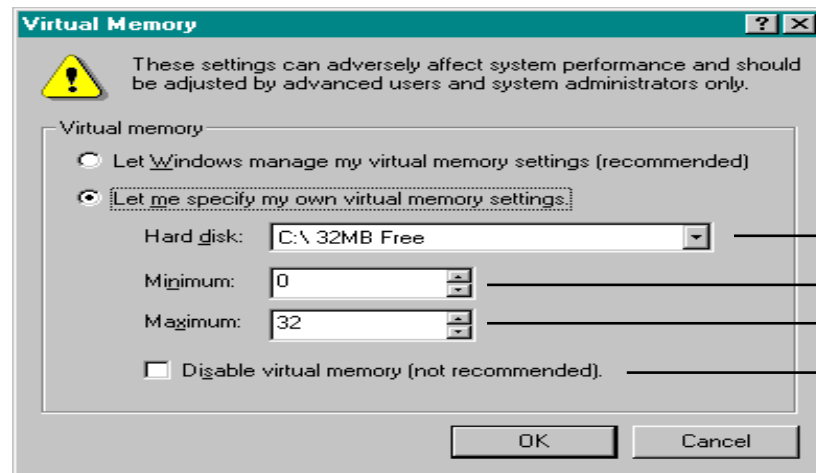
- **Field:** End Date, and Start Date are examples of fields.
- **Entry Source (Also known as input interface):** What dialog boxes can you use to enter data into this field? Can you import data into this field? Can data be calculated into this field? List every way to fill the field -- every screen, etc.

- **Display (Also known as output interface):** List every dialog box, error message window, etc., that can display the value of this field. When you re-enter a value into this field, will the new entry show up in each screen that displays the field? (Not always -- sometimes the program makes local copies of variables and fails to update them.)
- **Print (Also known as output interface):** List all the reports that print the value of this field (and any other functions that print the value).
- **Constrained By:** List every variable that constrains the value of this variable. (What if you enter a legal value into this variable, then change the value of a constraining variable to something that is incompatible with this variable's value?)
- **Constraint:** List every variable that this one constrains.

The “Let Windows manage my virtual memory settings” radio button constrains the availability of the features in the “Let me specify my own virtual memory settings” group.



Inactive Features



Active Features

# Test Methods and Test Case Design Techniques

## 4.1 Some Common Test Methods and Test Types

- Requirement Testing
- Exploratory Testing
- Smoke Testing
- Regression Testing

## 4.2 Test Case

- Test case criteria
- Test case essentials
- Test case syntax

## 4.3 Some Test Case Design Techniques

- Equivalent Partitioning
- Boundary Analysis
- Constraint Analysis
- **State Transition**
- Condition Combination



## State Transitioning

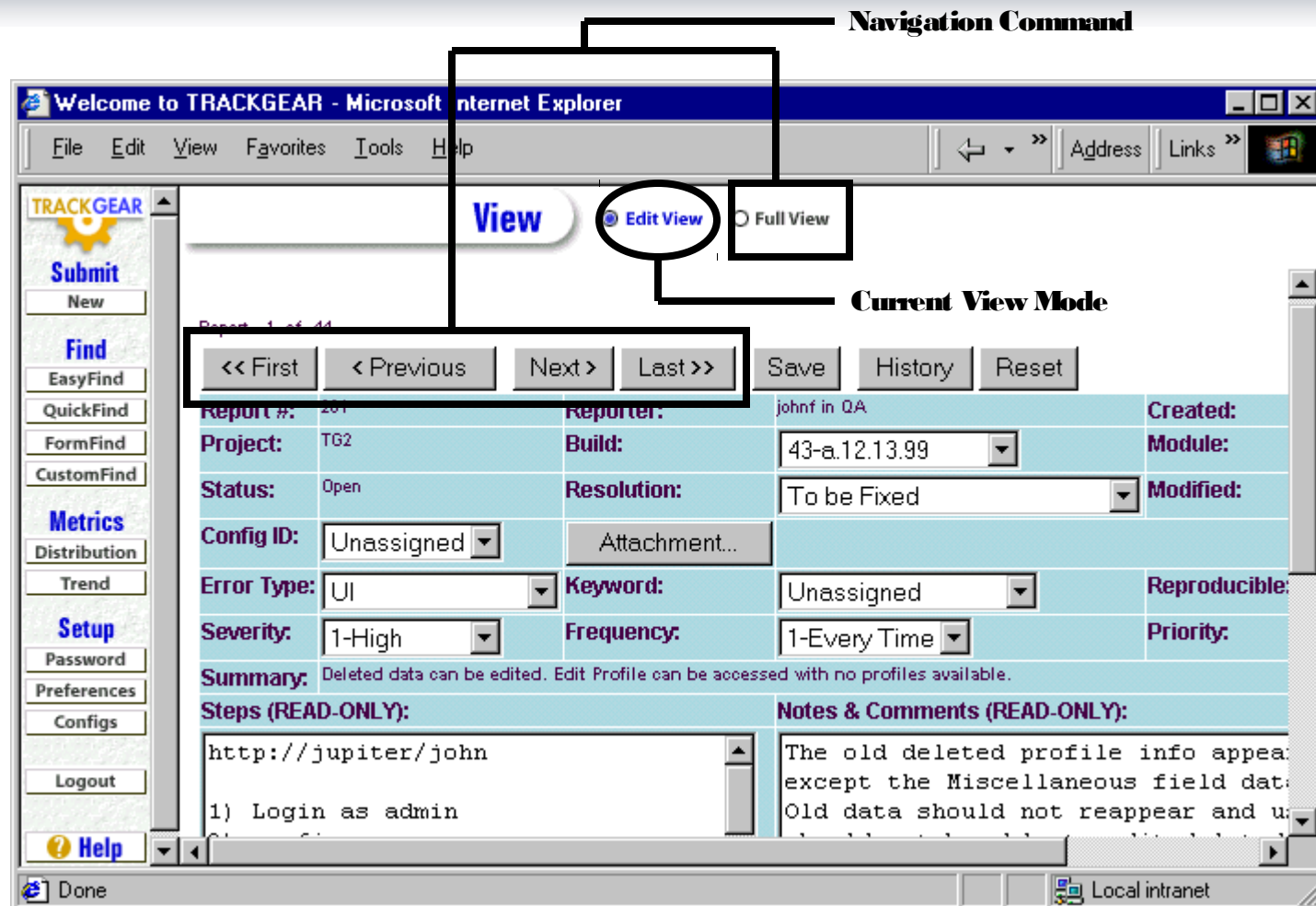
- Involves an analysis of the relationship among the states and the events or actions that cause the transitions from one state to another.

## General Steps

- Identify all supported states
- For each test define
  - **The starting state**
  - **The input events that cause the transitions**
  - **The output results of each transition**
  - **The end state**
- Tests should cover both positive and negative tests.

**Navigation Command**

**Current View Mode**



Report # 201

Project: TG2

Status: Open

Config ID: Unassigned

Error Type: UI

Severity: 1-High

Reporter: johnf in QA

Build: 43-a.12.13.99

Resolution: To be Fixed

Keyword: Unassigned

Frequency: 1-Every Time

Summary: Deleted data can be edited. Edit Profile can be accessed with no profiles available.

Steps (READ-ONLY):

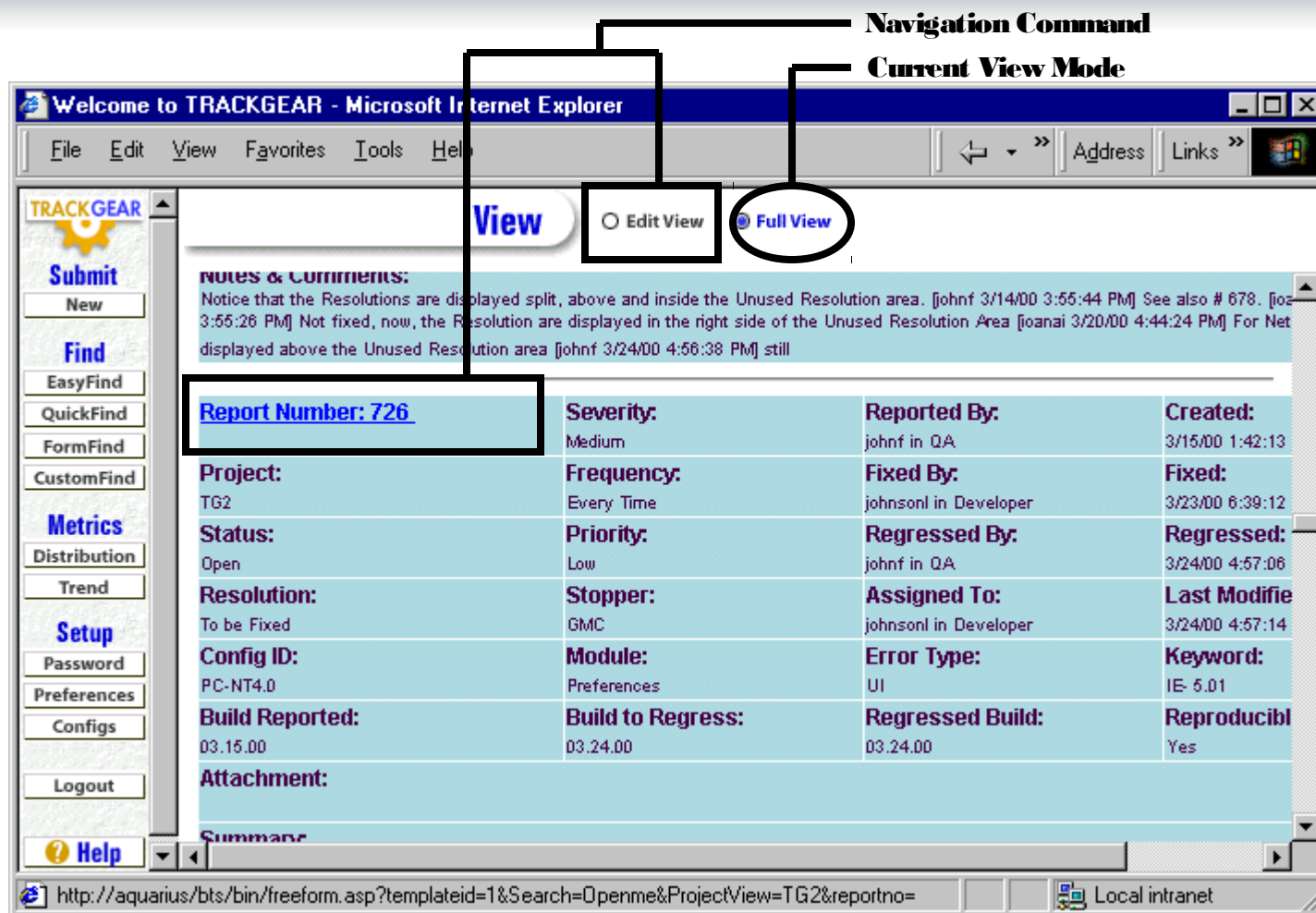
- http://jupiter/john
- 1) Login as admin

Notes & Comments (READ-ONLY):

The old deleted profile info appear except the Miscellaneous field data. Old data should not reappear and u...

**Navigation Command**

**Current View Mode**



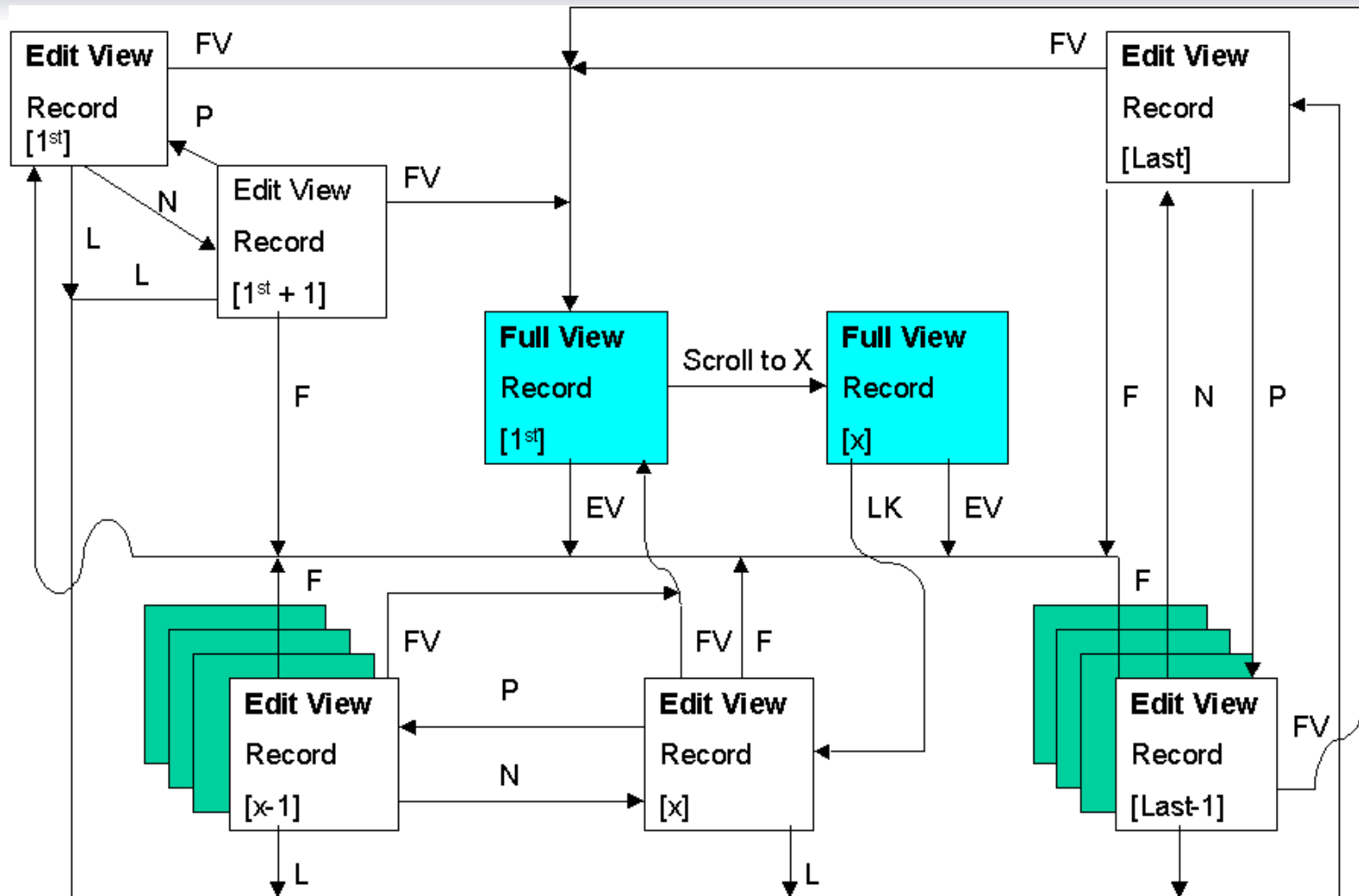
**View** ☐ Edit View ☒ Full View

**Report Number: 726**

<b>Project:</b> TG2	<b>Severity:</b> Medium	<b>Reported By:</b> johnf in QA	<b>Created:</b> 3/15/00 1:42:13
<b>Status:</b> Open	<b>Frequency:</b> Every Time	<b>Fixed By:</b> johnsonl in Developer	<b>Fixed:</b> 3/23/00 6:39:12
<b>Resolution:</b> To be Fixed	<b>Priority:</b> Low	<b>Regressed By:</b> johnf in QA	<b>Regressed:</b> 3/24/00 4:57:06
<b>Config ID:</b> PC-NT4.0	<b>Stopper:</b> GMC	<b>Assigned To:</b> johnsonl in Developer	<b>Last Modified:</b> 3/24/00 4:57:14
<b>Build Reported:</b> 03.15.00	<b>Module:</b> Preferences	<b>Error Type:</b> UI	<b>Keyword:</b> IE- 5.01
<b>Attachment:</b>	<b>Build to Regress:</b> 03.24.00	<b>Regressed Build:</b> 03.24.00	<b>Reproducible:</b> Yes

Summary

# Example 1



# Transitioning Example

## CHAPTER 4.3

Code VIEW MODE											NAVIGATION COMMAND					
0	Edit View-Record [1st]	Edit View displaying the 1st record								F	First					
1	Edit View-Record [1st+1]	Edit View displaying the 2nd record								P	Previous					
2	Edit View-Record [x]	Edit View displaying record [x]								N	Next					
3	Edit View-Record [x-1]	Edit View displaying record [x-1]								L	Last					
4	Edit View-Record [Last]	Edit View displaying the last record								FV	Full View					
5	Edit View-Record [Last-1]	Edit View displaying the next to last record														
6	Full View-Record [1st]	Full View displaying the 1st record								EV	Edit View					
7	Full View-Record [x]	Full View displaying record [x]								LK	Record ID Link					
Test Case No.		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Start View Mode		0	0	0	1	1	1	1	2	2	2	2	3	3	3	3
Navigation Command (Input)		N	L	FV	F	P	L	FV	F	P	L	FV	F	N	L	FV
End View Mode		1	4	6	0	0	4	6	0	3	4	6	0	2	4	6
Test Case No.		16	17	18	19	20	21	22	23	24	25					
Start View Mode		4	4	4	5	5	5	5	6	7	7					
Navigation Command (Input)		F	P	FV	F	N	L	FV	EV	EV	LK					
End View Mode		0	5	6	0	4	4	6	0	0	2					

# Test Methods and Test Case Design Techniques

## 4.1 Some Common Test Methods and Test Types

- Requirement Testing
- Exploratory Testing
- Smoke Testing
- Regression Testing

## 4.2 Test Case

- Test case criteria
- Test case essentials
- Test case syntax

## 4.3 Some Test Case Design Techniques

- Equivalent Partitioning
- Boundary Analysis
- Constraint Analysis
- State Transition
- **Condition Combination**

### Condition Combination

- Involves an analysis of the combination relationship of the variables such as browser settings. Each combination represents a condition to be tested with the same test script or procedure.

### General Steps

- Identify the variables.
- Identify the number of possible unique values of each variable.
- Create a table illustrating the complete unique combination of conditions formed by the variables and their values.

# Combination Test Case Design

**Combination Test Case Design**

Group A:

- ☒ Option1
- ☐ Option2
- ☐ Option3

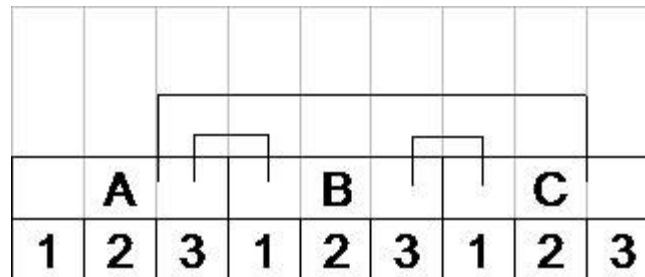
Group B:

- ☒ Option1
- ☐ Option2
- ☐ Option3

Group C:

- ☒ Option1
- ☐ Option2
- ☐ Option3

OK Cancel





# Test Design

Case	A	B	C
1	1	1	1
2	1	1	2
3	1	1	3
4	1	2	1
5	1	2	2
6	1	2	3
7	1	3	1
8	1	3	2
9	1	3	3

Case	A	B	C
10	2	1	1
11	2	1	2
12	2	1	3
13	2	2	1
14	2	2	2
15	2	2	3
16	2	3	1
17	2	3	2
18	2	3	3

Case	A	B	C
19	3	1	1
20	3	1	2
21	3	1	3
22	3	2	1
23	3	2	2
24	3	2	3
25	3	3	1
26	3	3	2
27	3	3	3

Application Under Test: Yahoo Mail

Example: We have a Test Requirement:

TR-ID	Test Requirements	Notes
TR001	User is able to login with valid username and password.	



Sign in to Yahoo!

Yahoo! ID:

Password:

☐ Remember my ID on this computer

[Sign In](#)

Why this is secure

[Forget your ID or password?](#) | [Help](#)

Don't have a Yahoo! ID?  
Signing up is easy. [Sign Up](#)

From that Test Requirement, we can design a simple Test Case as below.

TC-01: Login to Yahoo Mail with valid username/password.

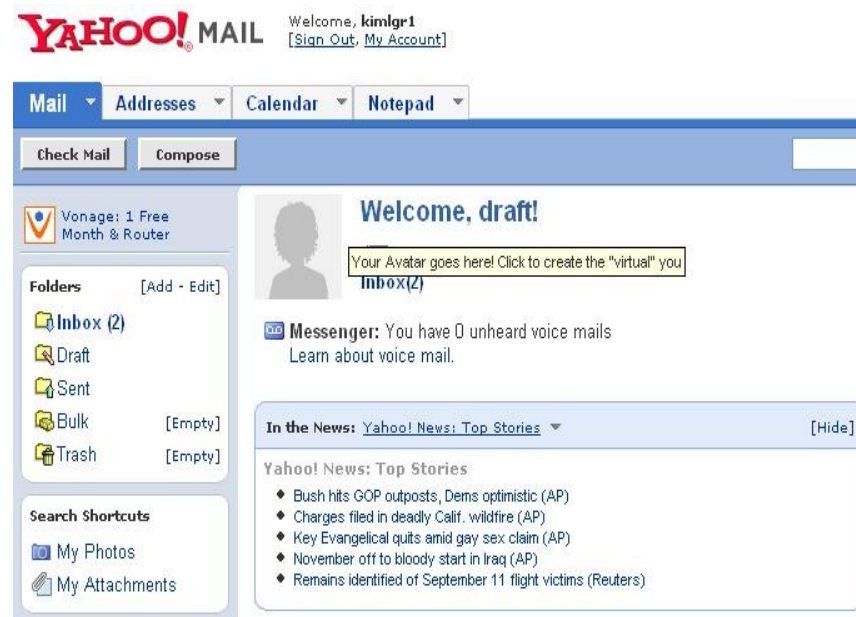
Step	Description	Data	Expected Result
1	Open IE and go to <a href="http://mail.yahoo.com">http://mail.yahoo.com</a>		Yahoo page is displayed.
2	Enter a valid yahoo ID into the 'Yahoo ID' text box	Yahoo ID: kimlgr1	
3	Enter a valid password into the 'Password' text box	Password: logigear	
4	Click on the 'Sign In' button		Login is successful.

- There are many methods to test an application: requirement-based testing, exploratory testing... The best practice is a combination of a few methods to satisfy the project needs.
- Test case is written to test an application. Some of the best techniques to design test case are equivalent partitioning and boundary analysis

### Application Under Test: Yahoo Mail

#### Exercise:

Please design a simple Test Case for one of the Test Requirements that you have created in the previous exercise.



- Write test case from test requirements:
  - Can save a file in MS after opening a new file
  - Can save a file in MS with text (use your name) and text with space, such as michealhackett and micheal hackett
  - Can save in the following format
    - XML
    - Web page
    - Plain text
    - Works 6



## **CHAPTER 5**

# **Working with a Test Case Management System**

## CHAPTER 5

- Test Case Tracking/Management System
- LogiGear® TestArchitect™ Test Case Management System
- 3<sup>rd</sup> party Test Case Management System
- Spreadsheet-based Test Case Management System

# OBJECTIVES

- What is a test case tracking/management system?
- An introduction to LogiGear® TestArchitect™ test case management system
- An introduction to a 3rd-party test case management system
- An introduction to spreadsheet-based test case management system





## CHAPTER 5.1

# Working with Test Case Management System

## 5.1 Test Case Tracking/Management System

5.2 LogiGear® TestArchitect™ Test Case Management System

5.3 3rd-party Test Case Management System

5.4 Spreadsheet-based Test Case Management System

# **Test Case Management System**

- Test case (TC) management system is the system to organize and control the process and artifacts required for the testing effort.
- The goal of TC management system is to allow teams to plan, develop, execute, and assess all testing activities within the overall software development effort. This includes coordinating efforts of all those involved in the testing effort, tracking dependencies and relationships among test assets and, most importantly, defining, measuring, and tracking quality goals.



# Test Management System Benefits

## CHAPTER 5.1

- Make the test process more transparent. QA and QC teams can manage their entire testing process.
- Ease of use
- Manage execution
- Manage change
- Easier test case maintenance
- Requirements traceability
- Various execution, result and performance
- Measurements
- Coverage Metrics
- Maximize time to market in test planning, execution, and results analysis.
- Management can monitor and examine the results and status of testing.
- Tests and results can be shared with vendors and customers.



## CHAPTER 5.2

# Working with Test Case Management System

5.1 Test Case Tracking/Management System

**5.2 LogiGear® TestArchitect™ Test Case Management System**

5.3 3rd-party Test Case Management System

5.4 Spreadsheet-based Test Case Management System



## CHAPTER 5.2

# LogiGear® TestArchitect™ Test Case Management System

- TestArchitect™ has the abilities of a test case management system:
  - Store test cases (in test modules)
  - Store test requirements
  - Store the results
  - Set up test plan and test suites to execute
- In which:
  - Test Module: An individual spreadsheet containing one or more *related* test cases.
  - Test Requirement: An *explicit* statement of the purpose of a test, based on product requirements or other quality criteria.
  - Test Case: A series of steps and verifications validating one or more test requirements

## CHAPTER 5.2

config		
<u>section</u>	<u>Test Objectives</u>	
test objective	TO_01	Verify access denied and show warning dialog when password is not matched
test objective	TO_02	Verify access is allowed when the username and password is correct
<u>section</u>	<u>Variable</u>	
	variable	value
set variable	mesg warning	Invalid Username or Password.
<u>section</u>	<u>Initialize</u>	
start application		
<b>TEST CASE</b>	<b>TC_01</b>	<b>Verify access is denied with invalid account</b>
test objective	TO_01	Verify access denied and show warning dialog when password is not matched
// Step 1	Login with invalid password	
log in	username	password
	john	logigear
// VP	Verify access denied and show warning dialog when username is not found	
check message	source	message
	warning messa...	# mesg warning
<b>TEST CASE</b>	<b>TC_02</b>	<b>Verify access allowed with valid account</b>
test objective	TO_02	Verify access is allowed when the username and password is correct
// Step 1	Login with username: john, password: <empty>	
log in	username	password
	john	
// VP	Verify access is allowed when the username and password is correct	
check window exists	window	welcome
<u>section</u>	<u>Exit App</u>	
close window	window	welcome



## CHAPTER 5.2

# TestArchitect™ Test Case Management System Pitfalls

- Not very clear
- Not easy to use



## CHAPTER 5.3

# Working with Test Case Management System

5.1 Test Case Tracking/Management System

5.2 LogiGear® TestArchitect™ Test Case Management System

**5.3 3rd-party Test Case Management System**

5.4 Spreadsheet-based Test Case Management System



# 3<sup>rd</sup>-party Test Case Management Systems

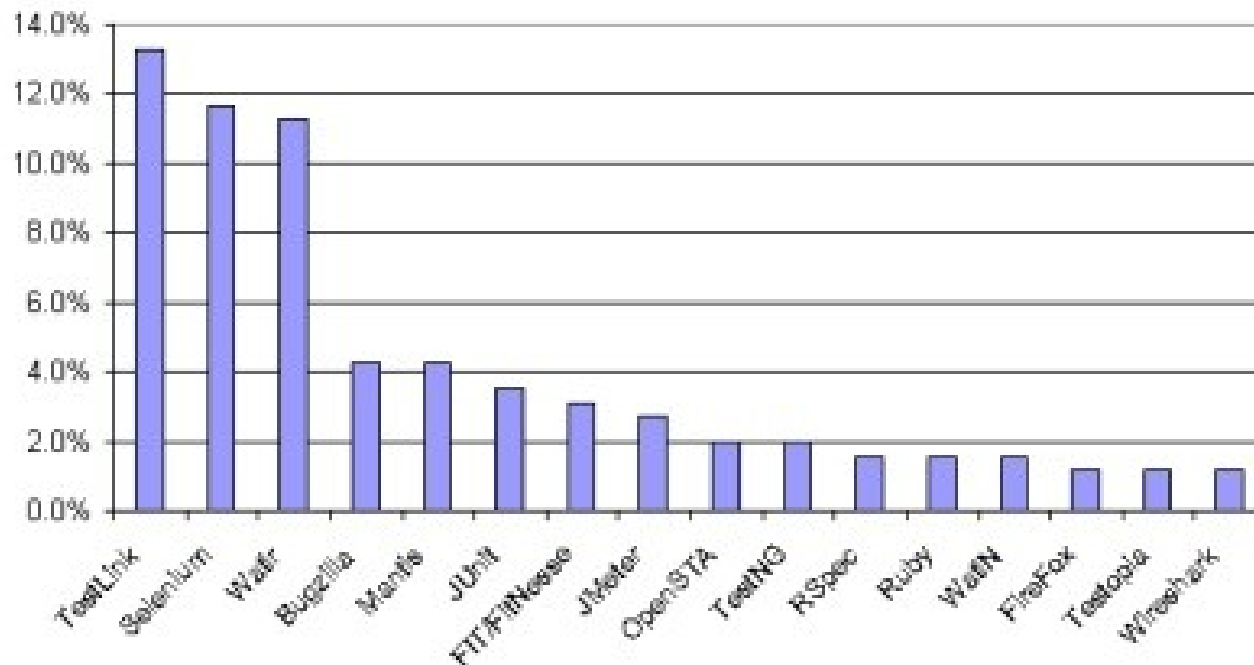
- There are many 3<sup>rd</sup> – party test management systems:
  - Test Link
  - QCIT (Quality Control Information Tool)
  - Test Director
  - Rational Test Manager Integrated with RequisitePro and ClearQuest
  - Empirex e-Manager: Web Testing Manager

## CHAPTER 5.3

- TestLink is web based Test Management system.
- Easily to create, manage Test cases and organize them into Test plans
- Test plans allow to execute Test cases and track test results dynamically, generate reports, trace software requirements, prioritize and assign tasks.
- The tool has web based interface with PHP and background database MySQL, Postgres or MS-SQL. It cooperates with known Bug tracking systems as is Bugzilla, Mantis, etc.
- TestLink is web based tool under the GPL license (free to use). The project is maintained by Open community of testers. Many developers on the team hold Quality Assurance Management positions and understand the needs of QA teams.
- TestLink makes Testing process easy and organized.

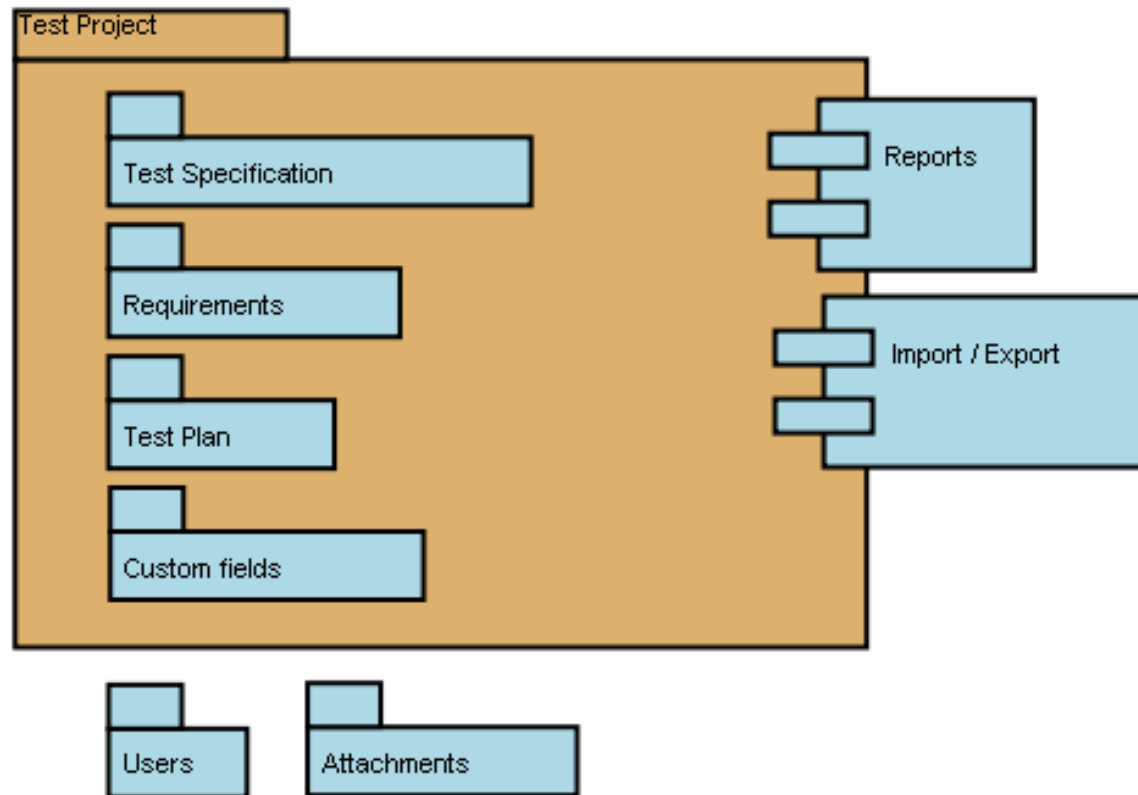
## CHAPTER 5.3

Tool popularity among survey respondents



- TestLink is the winning tool in [opensorcetesting.org](http://opensorcetesting.org) survey

# Test Link Overall Structure



## CHAPTER 5.3

- **Test Case** describes a testing task via steps (actions, scenario) and expected results. Test Cases are the fundamental piece of TestLink.
- **Test Suite** (Test Case Suite) organizes Test Cases to units. It structures Test Specification into logical parts.
- **Test Plan** is created when you'd like to execute Test Cases. Test Plans can be made up of the Test Cases from one or many Test Projects. Test Plan includes Builds, Milestones, user assignment and Test Results.
- **Test Project** is something that will exist forever in TestLink. Test Project will undergo many different versions throughout its lifetime. Test Project includes Test Specification with Test Cases, Requirements and Keywords. Users within the project have defined roles.
- **User**: each TestLink user has a Role that defines available TestLink features.

# Test Link Main Features

- **General Features:**
  - Functionality helps to follow standard testing processes as are specified by IEEE 829 or BCS SIGIST.
  - User web based GUI (Mozilla, Firefox, IE 6 compatible) help user access it without installation.
  - Multiple databases support TestLink runs with MySQL, PostgreSQL or MS-SQL.
  - Localization and Internacialization (English, French, German, Italian, Spanish, etc.)
  - Authentication support (internal or external LDAP) with user self-registration support
  - Flexible role based access control
  - Attachments and custom fields could be added to particular objects (for example Test Cases, Test results, etc.)
- **Test Projects:** Large team can divided the products to test projects. Each test project has own user rights, test specification, software requirements and Test plans.
  - Multiple projects support
  - Test cases are organized in hierarchal structure (tree menu) into Test Specification. History of each test case is traceable. Keywords are supported to bring more depth to test organization.
  - Users have defined roles in project (for example leader, tester, etc.)

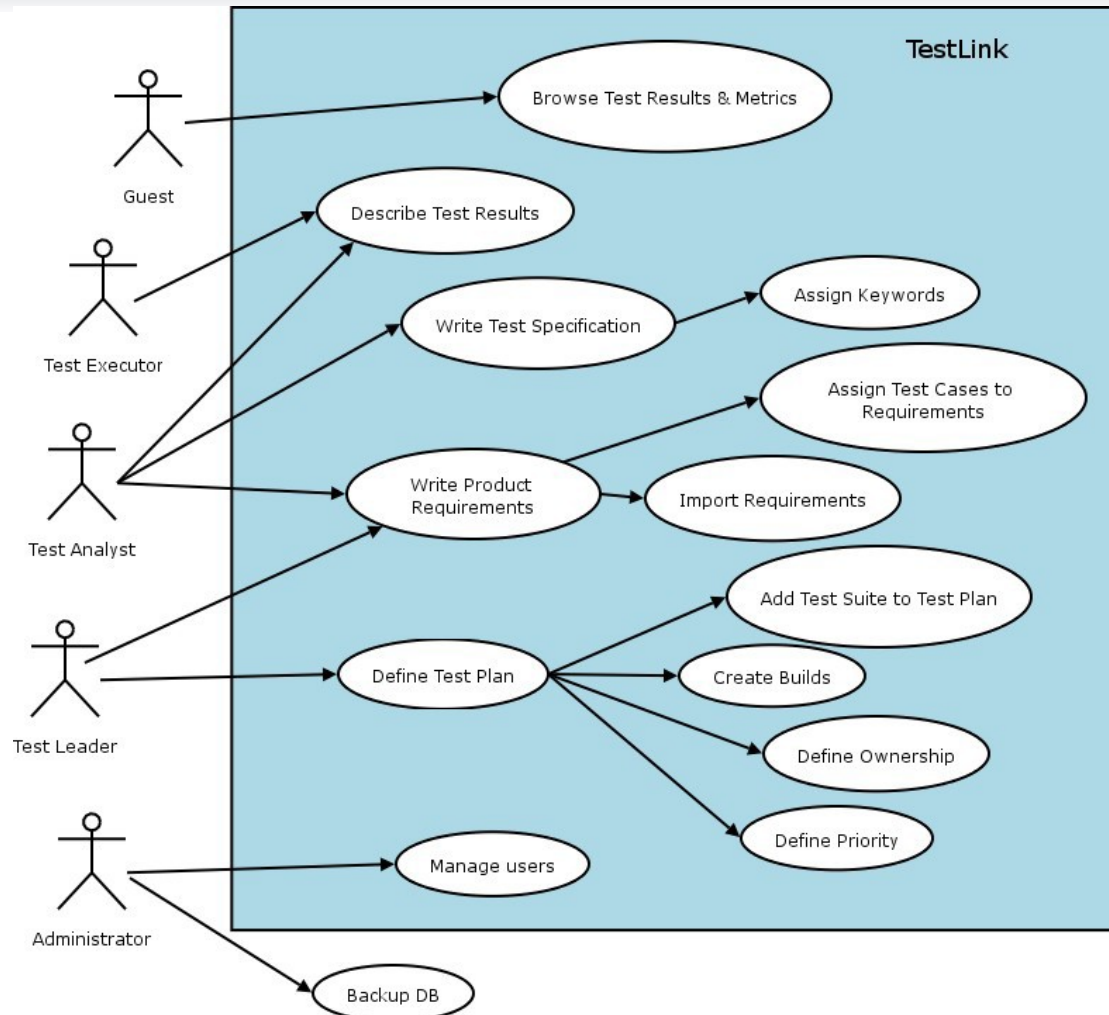


## CHAPTER 5.3

# Test Link Main Features

- **Requirements based testing:** Create or import own Software Requirement document and trace requirements to test results. Each test case can be mapped to a functional requirement. User can automatically generate Test Cases
- **Import/Export Test Cases:** Allows users to easily import and export test cases and product requirements and all of their test cases to and from XML file formats.
- **Test Plans:** Use TestLink to compose rich test plans containing an chosen set of test cases. Each Test Plans can collect test results for particular builds and platforms. Testing could be prioritized, assigned to testers, defined milestones. TestLink directly cooperates with Bug Tracking systems, i.e. Bugzilla, Mantis, Jira, TrackPlus, Eventum, Trac, Seapine, Redmine
- **Generate Reports**
  - Allows to view a variety of different test plan reports including: a bug report, a progress report, and failure rate report, and more.
  - Export of documentation to HTML, MS Word and MS Excel is supported.
  - User can also send reports directly via email.

# Test Link Functionality Overview





## CHAPTER 5.3

- <http://testlink.org/demo/index.php>



## CHAPTER 5.4

# Working with Test Case Management System

5.1 Test Case Tracking/Management System

5.2 LogiGear® TestArchitect™ Test Case Management System

5.3 3rd-party Test Case Management System

**5.4 Spreadsheet-based Test Case Management System**

# **Spreadsheet TC Management System**

- A spreadsheet is a computer application simulates a paper worksheet. It displays multiple cells that together make up a grid consisting of rows and columns, each cell containing either alphanumeric text or numeric values. A spreadsheet cell may alternatively contain a formula that defines how the contents of that cell is to be calculated from the contents of any other cell each time any cell is updated. Spreadsheets are frequently used for financial information because of their ability to re-calculate the entire sheet automatically after a change to a single cell is made
- LTRC usually uses Excel as a spreadsheet test case management system

## CHAPTER 5.4

Microsoft Excel - TrackGear 3.0 Test Cases.xls

File Edit View Insert Format Tools Data Window Help TestArchitect

100% Arial 10 B I U

	A	B	C	D	E
1			TrackGear Welcome/ Login (Special Test Cases)		
2					
3			Kim		
4			<b>Cases</b>	<b>Special Cases</b>	<b>Expected Results</b>
5					<b>Pass or Fail</b>
6			<b>Test User Name field</b>		
7			Submit form with invalid characters in the User Name field		
8	1.0		Access http://63.82.124.113/pleiku/		
9	1.1		Enter invalid characters in the User Name field (e.g. !@#\$%^&*()_~:;<?+/,")		
10	1.2		Enter valid password of that user in the Password field		
11	1.3		Click 'Login' button	If that user exists in the database, TG home page of user appears	
12					
13			Submit form with long text in the User Name field		
14	2.0		Access http://63.82.124.113/pleiku/		
15	2.1		Enter long text in the User Name field (e.g. a text of over 384 characters )		
16	2.2		Enter valid password of that user in the Password field		
17	2.3		Click 'Login' button	If that user exists in the database, TG home page of user appears	
18					
19			Submit form with special commands in the User Name field		
20	3.0		Access http://63.82.124.113/pleiku/		
21	3.1		Enter special commands in the User Name field (e.g. <HTML> or DELETE, DROP...)		
22	3.2		Enter valid password of that user in the Password field		
23	3.3		Click 'Login' button	If that user exists in the database, TG home page of user appears	
24					
25			Submit form with a bitmap content pasted into the User Name field from clipboard		
26	4.0		Access http://63.82.124.113/pleiku/		
27	4.1		Copy a bitmap pasted from clipboard to the User Name field		
28	4.2		Enter valid password of that user in the Password field		
29	4.3		Click 'Login' button	Error message displayed	

Special Cases / Easy Find Matrix for special / Easy find Matrix for functional / QuickFind

Ready

- There are many types of test case management system, here are 3 main types:
  - Test case managers that also interface with automation tools (TestArchitect™)
  - 3<sup>rd</sup> party test case manager
  - Spreadsheet-based test case manager
- Basing on the requirements of the projects, we choose the suitable test case management system to use

- Using TestLink to manage test cases of Mambo application



## CHAPTER 6

# Software Error

- What is a Software Error?
- Common Sources of Errors
- 13 Common Types of Software Errors
- Finding, Reproducing and Analyzing a Software Error
- Reporting a Software Error
- A Common Bug Life



# OBJECTIVES

- An introduction to software error, common sources of software error and some common types of software errors
- Learn to find, analyze bugs and write bug report
- An introduction to a common bug life

# Software Error

## **6.1 What is a software error?**

6.2 Common Sources of Errors

6.3 13 Common Types of Software Errors

6.4 Finding, Reproducing and Analyzing a Software Error

6.5 Reporting a Software Error

6.6 A Common Bug Life

# What is a Software Error?

## CHAPTER 6.1

- A software error is present when the program does not do what its user reasonably expects it to do.
- It is fair and reasonable to report any deviation from high quality as a software error.

***The existence of software errors reflects an impediment on the quality of the product, but does not necessarily imply that the developers are incompetent.***

# Software Error

6.1 What is a software error?

## **6.2 Common Sources of Errors**

6.3 13 Common Types of Software Errors

6.4 Finding, Reproducing and Analyzing a Software Error

6.5 Reporting a Software Error

6.6 A Common Bug Life

## CHAPTER 6.2

- You will report all of these types of problems, but it is important to keep straight in your mind, and on the bug report, which type you are reporting.
  - **Coding Error:** The program doesn't do what the programmer would expect it to do.
  - **Design Issue:** It's doing what the programmer intended, but a reasonable customer would be confused or unhappy with it.
  - **Requirements Issue:** The program is well designed and well implemented, but it won't meet one of the customer's requirements.
  - **Documentation / Code Mismatch:** Report this to the programmer (via a bug report) and to the writer (usually via a memo or a comment on the manuscript).
  - **Specification / Code Mismatch:** Sometimes the spec is right; sometimes the code is right and the spec should be changed.

# Software Error

6.1 What is a software error?

6.2 Common Sources of Errors

## **6.3 13 Common Types of Software Errors**

6.4 Finding, Reproducing and Analyzing a Software Error

6.5 Reporting a Software Error

6.6 A Common Bug Life



## CHAPTER 6.3

# 13 Common Types of Software Errors

- User Interface
- Error Handling
- Boundary-Related
- Calculation
- Initial and Later States
- Control Flow
- Handling or Interpreting Data
- Race Conditions
- Load Conditions
- Hardware/Environment Compatibility
- Source, Version, and ID Control
- Testing
- Documentation

Reference:

[http://www.logigear.com/resources/articles\\_lg/common\\_software\\_errors.asp](http://www.logigear.com/resources/articles_lg/common_software_errors.asp)

# Software Error

6.1 What is a software error?

6.2 Root Causes of Errors

6.3 13 Common Types of Software Errors

**6.4 Finding, Reproducing and Analyzing a Software Error**

6.5 Reporting a Software Error

6.6 A Common Bug Life



# **What to do in a bug finding process?**

1. Reproduce the error
2. Analyze the error
3. Report the error



## CHAPTER 6.4

# Reproducing a Software Error

- Some bugs are always reproducible, but some are just sometimes or even rarely.
- Bugs don't just miraculously happen and then go away. If a bug happens intermittently, it might be under some certain conditions.
- When we find a bug, we are looking at a *failure*, which is a set of symptoms of an underlying *error*.
- We hypothesize the cause, then we try to re-create the *conditions* that make the error visible.
- If the bug is non-reproducible, you should always report it, but describe your steps and observations precisely. Programmers will often figure them out.

# **Why is a Bug Hard to Reproduce?**

- Memory dependent
- Memory corruption
  - Run-time errors
  - Predicated on corrupted data
- Configuration dependent
  - Software
  - Hardware
- Timing related
- Initialization
- Data flow dependent
- Control flow dependent
- Error condition dependent
- Multi-threading dependent
- Special cases
  - Algorithm
  - Dates
- ...

# **Making an Error Reproducible**

- Write down everything you remember about what you did the first time.
- Note which things you are sure of and which are good guesses.
- Note what else you did before starting on the series of steps that led to this bug.
- Review similar problem reports you've come across before.
- Use tools such as capture/replay program, debugger, debug-logger, videotape, or monitoring utilities that can help you identify things that you did before running into the bug.
- Talk to the programmer and/or read the code.

# **Analyzing a Software Error**

## Why Analyze a Reproducible Bug?

- Analyze bugs in order to:
  - Make your communication effective;
    - Make sure you are reporting what you think you are reporting.
    - Make sure that questionable side issues are thoroughly investigated.
    - Create accountability.
  - Support the making of business decisions;
  - Avoid wasting the time of the programming and management staff;
  - Find more bugs.

# Analyzing a Reproducible Error

- Start by making sure the error is reproducible.
  - 1) Describe how to get the program into a known state. For example, starting the program takes it into a known (just booted) state.

Describe the state clearly, so that anyone familiar with the program will know what to do to get the program into that state.
  - 2) Specify an exact series of steps that expose the problem.
  - 3) Test your steps to make sure that you can reproduce the problem if you do exactly (and only) what it says in the bug report.

# Software Error

6.1 What is a software error?

6.2 Types of Errors

6.3 13 Common Types of Software Errors

6.4 Finding, Reproducing and Analyzing a Software Error

**6.5 Reporting a Software Error**

6.6 A Common Bug Life


# **How to Report a Software Error**


**Bug reports are your primary work product.**



# **A Useful Bug Report**

- Written (not reported orally)
- Uniquely numbered (ID required)
- Simple (non-compound - one bug per report)
- Understandable
- Reproducible
- Non-judgmental (only facts, no opinions)






**Submit**  
New

**Find**  
EasyFind  
QuickFind  
FormFind  
CustomFind

**Metrics**  
Distribution  
Trend

**Setup**  
User  
Project  
System  
Division  
Preferences  
Configs

Logout

 **Help**

**Submit New Report**

Save Save & Clone

<b>PROJECT:</b>	TGSample	<b>BUILD:</b>	1-alpha_01	<b>Module:</b>	Unassigned
<b>Config ID:</b>	Unassigned	Attachment...			
<b>Error Type:</b>	Compatibility-SW	<b>Keyword:</b>	Browser-NN4.x	<b>Reproducible:</b>	Yes
<b>Severity:</b>	2-Medium	<b>Frequency:</b>	2-Medium	<b>Priority:</b>	2-Medium

**SUMMARY:**  
Adding Project Member page does not refresh properly to show new members.

<b>STEPS:</b> 1. Log in as Admin. 2. Click the Project button in the Navigation bar. 3. Select the 'Project Members' radio button; click Go. 4. Add a new project member.	<b>Notes &amp; Comments:</b> Result: New member does not appear in the 'Project Members' list until the page is manually refreshed (F5).
---	---

**Assigned:** Auto Assigned
**Stopper:** 4-GMC

Save Save & Clone

# **Report Content**

- **Summary**
- **Description**
- **Steps to Reproduce – including expected behavior and observed behavior**
- Reproducible
- Severity
- Frequency
- Priority
- Keyword (Functional Area)
- Resolution

# Bug Summary

- This one-line description of the problem is the most important part of the report.
  - The project manager will use it when reviewing the list of bugs that haven't been fixed.
  - Executives will read it when reviewing the list of bugs that won't be fixed. They might only spend additional time on bugs with "interesting" summaries.
- The ideal summary tells the reader what the bug is, what caused the bug, what part of the program it's from and what its worst consequence is. It runs from 8 to 15 words long. You might not fit all this information in 15 words, but you might fit in the most important parts.
- We use the following syntax for writing the problem summary:

**Symptom + Action + Operating Condition**

# **Bug Summary**

## **Some good examples of Bug Summary:**

1. Run-time error when submitting the Contact Us form with first name of more than 256 characters.
2. The main dialog box is resizable
3. Help button does not bring up Help page.
4. Maximize button is still active while the dialog box is maximized.
5. Application crashed when clicking on the 'Submit' button in a Win2K system.
6. Top of 'Maria' image in 'Friends' page is not displayed.
7. Upgrade installation 1.0 fails if Window Media is running.

## **Some not-so-good examples of Bug Summary:**

1. Software fails
2. Can't install
3. Severe Performance Problems
4. Back button does not work
5. Cannot use an object with boolean expression
6. My browser crashed. I think I was on [www.foo.com](http://www.foo.com). I play golf with Bill Gates, so you better fix this problem, or I'll report you to him. By the way, your Back icon looks like a squashed rodent. Too ugly. And my grandmother's home page is all messed up in your browser.

# **Bug Description and Steps to Reproduce**

- First, describe the problem. What is the bug? Don't rely on the summary to do this – a short line sometimes cannot state all what you want to say.
- Next, go through the steps that you use to recreate this bug. Start from a known place (e.g. boot the program) and then describe each step until you hit the bug.
- Describe the erroneous behavior and if necessary, explain what should have happened. (Why is this a bug? Be clear.)
- If you expect the reader to have any trouble reproducing the bug (special circumstances are required), be clear about them.

- A clear, reproducible set of steps may be all you need to file the report. But you may be able to improve the report in four ways:
  - 1) You might be able to *simplify* the report by **eliminating unnecessary or irrelevant steps**.
  - 2) You might be able to *simplify* the report by **splitting it into two reports**.
  - 3) You might be able to *strengthen* the report by **showing that it is more serious than it first appears**.
  - 4) You might be able to *strengthen* the report by **showing that it is more general than it first appears**.



# 1. Eliminate Unnecessary Steps

- Sometimes it is not immediately obvious what steps can be dropped from a long sequence of steps in a bug.

***Look for critical steps -- Sometimes the first symptoms of an error are subtle.***

- You made a list of all the steps that you took to show the error. You are now trying to shorten the list.
  - If you've found what looks like a critical step, try to eliminate almost everything else from the bug report. Go directly from that step to the last one (or few) that shows the bug.
  - In general, try taking out individual steps or small groups of steps. Can you show the bug with the others?

## 2. Split the Report in Two

- When you see two related problems, you *might* report them together on the same report as long as you show that there are two of them.
- But if this lengthens the report or makes it at all confusing, write two reports instead. If you would have to run two tests to verify that a bug has been fixed, it's fair to report the two symptoms that you have to test on two separate bug reports.
- When you report related problems, it's a courtesy to cross-reference them. For example:

Related bug -- see Report # xxx

## 3. Show that it is More Serious

- **Look for follow-up errors:**
  - Keep using the program after you get this problem. Does anything else happen? Sometimes a modest-looking bug can lead to a system crash or corrupted data.
- **Look for nastier variants:**
  - Vary the conditions under which you got the bug. For example, try to get the bug while the program is doing a background save. Does that cause data loss or corruption along with this failure?
- **Look for nastier configurations:**
  - Sometimes a bug will show itself as more serious if you run the program with less memory, a higher resolution output, more graphics on a page, etc.

## 4. Show that it is More General

- **Look for alternative paths to the same problem:**
  - Sometimes the bug can happen in some alternative path.
  - For example, a bug that happens when deleting a file can happen when deleting a folder also.
- **Look for configuration dependence:**
  - Sometimes the bug happens because of some hardware configuration.
  - For example, a graphic/UI bug might have some tie to the graphic card driver, the graphic card manufacturer or the screen resolution.

# Reproducible

- You may or may not have this on your form, but you should always provide this information.
  - Never say it is reproducible unless you have recreated the bug. (Always try to recreate the bug before writing the report.)
  - If you have tried and tried but you can not recreate the bug, say No. Then explain what steps you tried in your attempt to recreate it.
  - If the bug appears sporadically and you do not yet know why, say “sometimes” and explain.

## CHAPTER 6.5

- You will have to rate the bug's seriousness. Many companies use a three-level rating:
  - 1 - Critical: This means fatal to the release (unacceptable to ship)
  - 2 - Serious: It's a bad bug, but it doesn't, for example, cause data loss or a program crash.
  - 3 - Minor: It's a bug, but it's not a big deal.
- Your company's definitions may be a bit different from these. And you might have a five-level rating instead of three -- check with your manager.
- Many companies sort their summary reports by severity, so you want to fill in this field thoughtfully.

- With some bug tracking systems, you also have to rate the bug frequency. **Frequency** is usually graded by assessing the following three characteristics:
  - How easy is it for the user to encounter the bug
  - How frequent would the user encounter the bug
  - How often the buggy feature is used
- Many companies use a three-level rating:
  - 1 - Always
  - 2 - Often
  - 3 - Seldom

- **Priority** rating is either automatically generated by the bug tracking system by assessing the Severity and the Frequency ratings or assigned only by the project manager. This is the *fix-priority* that everyone who is responsible for working on the bug will go by.



# **Keyword (Functional Area)**

- You may have to categorize the bug according to its functional area.
- The tracking system should include a list of the possible keywords. Read the list and ask questions in order to learn what each category includes.
- It is important to categorize these bugs consistently. They'll often be assigned out and summary-reported by category. Burying a bug in the wrong category can lead to its never getting fixed.
- If you're creating the list of functional areas, keep it short, perhaps 20 areas.

## CHAPTER 6.5

- The project manager has the privilege to assign most of the resolutions in this field.
- Common resolutions include:
  - **New:** The newly submitted bug
  - **To Be Distributed:** The bug is waiting to be distributed
  - **To Be Fixed:** The bug is being fixed.
  - **QA Info Request:** The bug needs more clarification from Tester.
  - **Developer Info Request:** The bug needs more clarification from Developer.
  - **Not Reproducible:** The bug cannot be reproduced.
  - **Fixed:** The bug is fixed.
  - **Not a Problem:** The application works as it is supposed to.
  - **Duplicate:** The bug is just a repeat of another bug.
  - **Deferred:** The bug will be fixed in a later release.
  - **Feature Limitation:** There is some feature limitations that do not allow to fix the bug.

# Bug Report

## Simple Template

**Bug ID:** 001

**Summary**

**Main Form:** The application does not close when clicking on End button

**Description:** The application can be closed when clicking on X button but cannot be closed when clicking on End button

**Steps:**

1. Open MiniBank
2. Click on End button
3. Observe the result

**Expected Result:** The application is closed

**Observed Result:** The application is still on the screen

# Software Error

6.1 What is a software error?

6.2 Types of Errors

6.3 13 Common Types of Software Errors

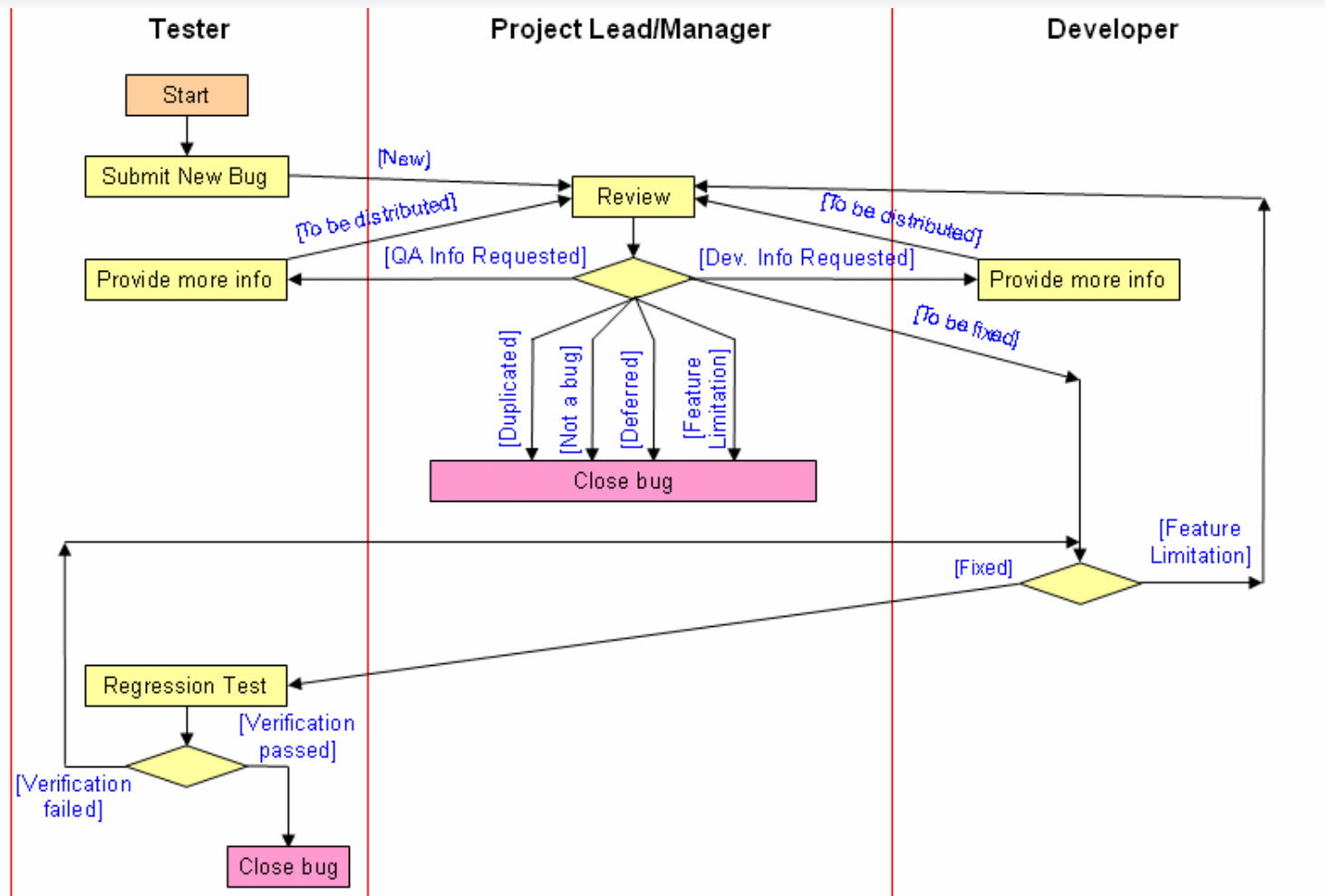
6.4 Finding, Reproducing and Analyzing a Software Error

6.5 Reporting a Software Error

**6.6 A Common Bug Life**

# A Common Bug Life

## CHAPTER 6.6



- A software error is present when the program does not do what its user reasonably expects it to do.
- Software Errors usually come from 5 common sources: Coding Error, Design Issue, Requirement Issue, Documentation/Code Mismatch, Specification/Code Mismatch
- There are a lot bugs in a software, however, they are belong to 13 types of software errors.
- Bug report is the main product of tester. To write a bug report, the tester has to find, reproduce, analyze and finally report the bugs.



## CHAPTER 6

# EXERCISE



## CHAPTER 7

# Working with a Bug Management System





# OUTLINE

## CHAPTER 7

- Bug Tracking/Management System
- LogiGear® TrackGear® Bug Management System
- Reporting Bugs Using TrackGear®



## CHAPTER 7

# OBJECTIVES

- An introduction to a bug tracking/management system
- An introduction to LogiGear® TrackGear® bug management system
- Learn to report bugs using TrackGear®



**CHAPTER 7.1**

# **Working with a Bug Management System**

## **7.1 Bug Tracking/Management System**

7.2 LogiGear® TrackGear® Bug Management System

7.3 Reporting Bugs Using TrackGear®

- A **bug tracking/management system** is a software application designed to help QA and programmers keep track of reported software bugs in their work.
- Many bug-tracking systems used by most open source software projects, allow users to enter bug reports directly. Other systems are used only internally in a company doing software development. Typically bug tracking systems are integrated with other software project management applications.
- Having a bug tracking system is extremely valuable in software development, and they are used extensively by companies developing software products

- The major component is a **database** records facts about known bugs. Facts may include the time a bug was reported, its severity, the erroneous program behavior, the details on how to reproduce the bug, the identity of the person who reported it and any programmers who may be working on fixing it...
- Typical bug tracking systems support the concept of the life cycle for a bug which is tracked through status assigned to the bug. A bug tracking system allow administrators to configure permissions based on status, move the bug to another status, or delete the bug. The system also allow administrators to configure the bug statuses and to what status a bug in a particular status can be moved to.

# Bug Tracking Usage

- A bug-tracking system **may be** used to on the productivity of programmers at fixing bugs (1).
- A *local bug tracker (LBT)* is usually a computer program used by a team of application support professionals to keep track of issues communicated to software developers. Using an LBT allows:
  - Support professionals to track bugs in their "own language" and not the "language of the developers."
  - A team of support professionals to track specific information about users who have called to complain that may not always be needed in the actual development queue (thus, there are two tracking systems when an LBT is in place.)



CHAPTER 7.2

# Working with a Bug Management System

7.1 Bug Tracking/Management System

**7.2 LogiGear® TrackGear® Bug Management System**

7.3 Reporting Bugs Using TrackGear®



## CHAPTER 7.2

# LogiGear® TrackGear® Bug Management System

- A Web-based bug tracking system that is powerful, flexible, and simple to use.
- Offers custom workflow, e-mail notification, meaningful metrics and attachment features.
- Supports multiple projects while maintaining security at user, group and project levels.
- Create unique queries with flexible search tools. Intuitive interface - access any information with just a few clicks. Supports any client platform with Netscape Navigator 4.x or MS Explorer 4.x.



## CHAPTER 7.2

- **Analyze project status** quickly with an unlimited number of customizable report formats.
- **Interpret bug statistics** with predefined metrics and present information in meaningful charts and reports. Configure TRACKGEAR's workflow to fit bug-resolution processes.
- **Enhance workflow** with features like auto-routing and user-defined email notification.
- **Enforce security** at the user level, group level, and project level simultaneously.
- **Scale TRACKGEAR** to support multiple projects in the same or different divisions all at the same time! Attach multiple files to a single report for easy downloading later.



# Flexible TrackGear

## CHAPTER 7.2

- **Find bug reports** quickly with TRACKGEAR's powerful search tools: Easy Find, Quick Find, and Form Find. Or, use Custom Find to create your own unique queries.
- **Share customized report formats** and queries with other users-or keep them private.
- **Utilize TRACKGEAR's remarkable** accessibility and speed to work from anywhere you use the Web: at the office, at home, or on the road.
- **Support different divisions** within a company in customizing TRACKGEAR to meet individual needs; each division can have a unique system design and custom workflow!
- **Communicate critical information** to users with admin broadcast messaging-announce scheduled maintenance shutdowns hours in advance.
- **Set workflow rules** to reflect existing or new processes. TRACKGEAR will automatically enforce them!
- **Support any number** of individual user accounts-and purchase additional concurrent user licenses as your company grows.



## CHAPTER 7.2

# Simple TrackGear

- **Access** any TRACKGEAR feature with just a few clicks.
- **Navigate** with TRACKGEAR's intuitive user interface-no wasted time learning a new process.
- **View reports** in one of three intuitive layouts; change views instantly as your needs and tasks change.
- **Get your product team started** right away-with TRACKGEAR's easy setup and installation process you'll be up and running in no time!



# Other Features

## CHAPTER 7.2

- **Customize** essential system attributes and optimize your company's workflow-Resolution, Priority, Keyword, Error Type and more!
- **Trace any report's history** with automatic signature and date stamping for reliable auditing and user accountability.
- **Set preferences** for user-defined start pages-each user starts their session from the TRACKGEAR screen that is most relevant to them.
- **Construct and share** multiple, complex search queries to automate repetitive search tasks.
- **Create unique profiles** for numerous hardware/software configurations and save time reentering identical information.
- **Save data-entry time** by cloning similar reports.



CHAPTER 7.3

# Working with a Bug Management System

7.1 Bug Tracking/Management System

7.2 LogiGear® TrackGear® Bug Management System

**7.3 Reporting Bugs Using TrackGear®**

# Reporting Bugs Using TrackGear®

- Default new report form

Save
Save & Clone
Spell Check

PROJECT:
TG-Sample
BUILD:
Alpha01
Module:
Unassigned

Summary:

Steps:

Notes & Comments:

Config ID:
Unassigned
Cross References

URLs
Attachments

Associate Source Files

Assigned:
Auto Assigned
☐ Private Report

Save
Save & Clone
Spell Check

# Explanations for the Various Fields

- **Project:** This will have the different projects the user is assigned to. User select a project to log-in a defect.
- **Build:** If a particular project has multiple builds, user can select the same.
- **Module:** This will help us to select the modules of the application for which the bug/issue is going to logged into
- **Summary:** Give a general description of the bug/Issue.
- **Steps:** Specify the detailed steps to reproduce the bug.
- **Assigned:** Assign the defect/issue to someone may concern.
- **Due Date:** Mention the date by which you expect the solution.
- **Attachments:** To attach the screen shots of the bug or any materials which you think it can help the bug is clearer
- **Save:** Click on “Save” button to complete logging in the bug.
- Once the bug/issue is saved, a new screen for logging in new bugs/issues will be displayed.
- And some other fields...

## CHAPTER 7.3

- A completed bug report:

Total Reports: 1

<b>Report Number:</b> <u>1</u>	<b>Reporter:</b> tg in Developer	<b>Created:</b> 11/14/2001 1:00:04 PM
<b>Project:</b> TG-Sample	<b>Build Reported:</b> Alpha01	<b>Module:</b> Search
<b>Status:</b> Open	<b>Resolution:</b> New	<b>Last Modified:</b> 11/14/2001 1:00:04 PM
<b>Summary:</b> Search results are incomplete		
<b>Steps:</b> 1. Enter site 2. Click Search tab 3. Click Search Now button		
<b>New Notes &amp; Comments:</b> Note that all available results aren't returned.		
<b>Assigned To:</b> tg in QA		
<b>Fixed By:</b>	<b>Build to Regress:</b>	<b>Fixed:</b>
<b>Regressed By:</b>	<b>Regressed Build:</b>	<b>Regressed:</b>
<b>Reference:</b>		
<b>Attachment:</b>		<b>Url:</b>



## CHAPTER 7

- A **bug tracking/management system** is a software application designed to help QA, QC, testers and programmers keep track of reported software bugs in their work.
- Bug tracking systems are extremely valuable in software development, and they are used by companies developing software products
- The major component is a **database** records facts about known bugs.
- In LogiGear, we use TrackGear for keeping track and managing bugs. It is a web-based bug tracking system that is powerful, flexible, and simple to use.

- Using TrackGear to track bugs of Mambo application



## CHAPTER 8

# Reporting Progress: Status Report

## CHAPTER 8

- Reporting Test Progress
- Goals of Status Report
- Sample Template of a Status Report

# **OBJECTIVES**

- An introduction to testing status report and its goals
- Learn about how to write a status report



## CHAPTER 8.1

# Reporting Progress: Status Report

### 8.1 Reporting Test Progress

### 8.2 Goals of Status Report

### 8.3 Sample Template of a Status Report

- These reports need to convey to the team:
  - What has been tested
  - What has not been tested
  - What is in-test
  - What will be tested in the next period (week, phase, build)
  - Test results
  - Test team needs and issues
  - Overall progress of the assignments
  - Any item, problem, etc. that is holding the team back from testing

# Reporting Test Progress

## CHAPTER 8.1

- This is our communication tool to the rest of the team.
- Tests or testing that is: blocked, failed, skipped
- As with all writing – write to the audience. This report may go out to non-technical staff. Results should be reported in terms of: schedule, budgets, features and quality. Not core-dumps or uninitialized pointers. Also – keep the tone non-judgmental
- Be calm! Speak in numbers and trends.
- Reporting has to be consistent.



- If you are a Test Lead, you have to write Weekly Status or Milestone Reports. They are very important for tracking and prompting change. Many times on a development project changes are approved and not documented. Your report may be the only place these changes are captured. Important items from status reports can be rolled into Test Plan revisions
- Types of Test Status Report:
  - Test Coverage Analysis, Bug Summery Report, Test Suite Summery, Test Case Progression



## CHAPTER 8.2

# Reporting Progress: Status Report

8.1 Reporting Test Progress

**8.2 Goals of Status Report**

8.3 Sample Template of a Status Report

# Goals of Status Report

## CHAPTER 8.2

- The Goal is communicating Test Progress
- This takes out wondering what the test group is doing.
- It gives test assurance of what is in test and what is not tested
- It can be just defects found, defects fixed – if nothing more than that it is a start



## CHAPTER 8.3

# Reporting Progress: Status Report

8.1 Reporting Test Progress

8.2 Goals of Status Report

**8.3 Template of a Status Report**



# Status Report Example

## CHAPTER 8.3

### SafetyNet 2008.1.MS0 & Blackberry 1.2.16931 2009.1.MS0 Bold - Testing Status Report

#### Today's Results:

##### TODAY'S TASKS:

- Testing the <http://social2.4thmedia.com/social-web> by using FF2 browser -  
**Done**
- Testing the <http://social2.4thmedia.com/pc-web/b.action?t=84ca18666b> and  
verify can sign up a PIMPed user through this site - **Done**

##### Unusual Issues for the day:

- None

##### Blocking Bugs: 1

- Q [SNT-9334] - JspException occurs on server side and blank page displays when  
going to Inspection Report of any inspection with at least 1 observation

##### Re-open issues: 1

- Q [SNT-9091] - watchListGraph ArrayIndexOutOfBoundsException from Dashboard

##### New Bugs found today: 2

- Q [SNT-9459] - Y Axis is cut-off on Dashboard
- Q [SNT-9457] - Time in "Report updated..." text at the bottom of Reports page  
is inconsistent with the time of Last Inspection updated

##### Automation Run Schedule:

- Our working time today during your night was: 5:30PM - 4:30AM
- Schedule to run automation during your daytime - **January 16:** We are running  
regression automation test for Dev Safety, Safety Demo and JE Dunn zones on  
this build 2009.1 MS0.

##### Outstanding Issues:

- None.

##### Number of bugs regressed for Today: 2



# Status Report Example (cont.)

## CHAPTER 8.3

### Overall Progress:

#### Manual Test:

- **Regression Test** (Total number of regressed issues for all Web app and Devices app): 39
- **Exploratory Testing:**

#### **1. Web:**

Screen Resolution: 1280 x 1024

#### **On IE7 Browser (All features)**

- + Dev Safety Zone: Explored 100% (7/7 features)
- + DPR Zone: Explored 100% (11/11 features)
- + Dev Combine Zone: Explored 100% (7/7 features)
- + Safety Demo Zone: Explored 100% (8/8 features)

#### **2. Device (Inspections feature):**

##### **A. Pocket PC:**

- + Dev Safety Zone: Explored 100%
- + DPR Zone: Explored 100%
- + Turner Zone: Explored 100%

Total new issues of both Web and Pocket PC on this current build

- Number of new bugs: 47
- Number of re-open issue: 5
- Number of improvement issue: 0

#### Automation Test

The status of running regression test on the following zones for build 2009.1.MS0:

Zones	Automation Regression Status	Notes
Dev Safety	In Progress	
Safety Demo	In Progress	
Dev Combine	Done	
Turner	In Progress	Test module "Edit Company - Inspections" is blocked due to [SNT-9334]  Test module "Edit Contact - Inspections" is blocked due to [SNT-9334]  25% of test Module "User Roles - Sync-only" is blocked due to [SNT-9334]

# **Status Report**

- With every issue, you should suggest/restate the solution
- Outstanding issue is the one can affect your all tasks or it may occur for a long time
- Issue for the day is the one that only occur on the day you report
- With every assignment, you should restate the percentage of completion, status of the tasks
- ...

## CHAPTER 8

- Status report is used to communicate test result and testing progress.
- Status report can include so many things depend on the testing project is being handled. Main things usually mentioned in status report are issues, assignments, resources, test progress, test results...
- Testers write status reports everyday and send them to their test lead. The leader will collect status reports of all team members and send a status report of the team to QA





**Thank you!**