

Module FNDA – Fundamentals of Testing

Lesson 1 – What is Testing?	1
Lesson Objectives	1
The Software Risk.....	2
The Impact of Software Failure	3
The Goal of Software Testing.....	4
The ISTQB Definition of Software Testing.....	4
Isn't Software Testing Just About Running Tests?.....	5
Testing Is A Process.....	5
Testing Activities	6
Static & Dynamic Testing.....	6
Verification and Validation.....	7
Objectives of Software Testing	7
Objectives of Software Testing (2)	8
Varying Test Objectives	8
Examples.....	9
Testing vs. Debugging	9
Testing vs. Debugging Example	10
The Role of Testing in a Project	11
Lesson 2 – Why is Testing Necessary?.....	12
Learning Objectives	12
Why is Testing Needed?.....	13
Testing's Contribution to Success	14
Examples.....	15
Examples (2)	15
The Role of Test Objectives	16
Testing is Not QA!.....	16
Quality Assurance.....	17
The Role Of Root Cause Analysis	17
Quality Control	18
QA and QC Example	18
Why Software Fails.....	19
A Few Causes of Software Failure	20
The First “Computer Bug”.....	22
Not All Unexpected Test Results Are Failures.....	23
False Positives and False Negatives	24
False Positives and False Negatives	25
Root Causes	25
Example	26
In This Example.....	26
Important Terms.....	27
Lesson 3 – General Testing Principles.....	28
Lesson Objectives	28
General Testing Principles.....	28
Principle 1 – Testing Shows The Presence Of Defects, Not Their Absence	29
Principle 2 – Exhaustive Testing Is Impossible	30
Principle 3 – Early Testing Is Good.....	31
Principle 4 – Defects Tend To Cluster.....	32
Principle 5 – Beware of The Pesticide Paradox.....	33
Principle 6 – Testing Is Context Dependent	34
Principle 7 – The Absence-Of-Errors Fallacy	35
Lesson 4 – Testing Processes.....	36
Lesson Objectives	36
What is a Test Process?.....	36
Factors That Influence the Testing Process	37
Deconstructing the Test Process	38
Test Coverage	39

The Test Basis.....	40
Example of Context Dependencies.....	40
Fundamental Testing Process.....	41
Example.....	42
Fundamental Testing Processes.....	43
What is Testware?.....	44
Test Planning.....	44
Test Monitoring and Control.....	45
Examples.....	45
Reporting Test Progress.....	46
Test Analysis.....	47
Test Analysis Tasks (2).....	48
Test Analysis Tasks (3).....	48
Test Analysis Tasks (4).....	49
Test Analysis Tasks (5).....	49
Application of Test Techniques In Test Analysis.....	50
Defect Identification During Test Analysis.....	51
Example.....	51
Test Design.....	52
Test Design Activities.....	52
Going From Test Conditions to Test Design.....	53
Test Implementation.....	53
Test Completion Activities.....	58
Lesson 5 – Work Products Used in Testing.....	59
Learning Objective.....	59
Variations and Guidelines.....	59
Test Planning Work Products.....	60
Test Monitoring And Control Work Products.....	60
Test Monitoring And Control Work Products (2).....	61
Test Analysis Work Products.....	61
Test Design Work Products.....	62
Test Design Work Products (2).....	62
Test Implementation Work Products.....	63
Test Implementation Work Products (2).....	64
Test Implementation Work Products - Test Data.....	64
Exploratory Testing Work Products.....	65
Test Execution Work Products.....	65
After Test Execution is Complete.....	66
Test Completion Work Products.....	66
Example.....	67
Traceability Between The Test Basis And Test Work Products.....	67
Benefits of Good Traceability.....	68
Benefits of Good Traceability (2).....	68
Tool Support for Test Work Products.....	69
Example Traceability Matrix.....	69
Lesson 6 – The Psychology of Testing.....	70
Objectives.....	70
The Psychology of Testing.....	70
Communicating About Failures.....	71
Confirmation Bias.....	72
Blaming the Bearer of Bad News.....	72
Perceptions of Testing.....	73
Good Communication Skills.....	74
Ways to Communicate Well.....	75
Ways to Communicate Well (2).....	75
Clear Test Objectives.....	76
Tester’s and Developer’s Mindsets.....	76
The Tester’s Mindset.....	77
The Developer’s Mindset.....	78

Example of Mindset Differences	79
Example of Mindset Differences (2).....	80
To Learn More About the Human Issues in Testing.....	80



FNDA

The Fundamentals of Testing

Objectives

At the end of this module, you should be able to:

- Identify typical objectives of testing (K1)
- Differentiate testing from debugging (K2)
- Give examples of why testing is necessary (K2)
- Describe the relationship between testing and quality assurance and give examples of how testing contributes to higher quality (K2)
- Distinguish between error, defect, and failure (K2)
- Distinguish between the root cause of a defect and its effects
- Explain the seven testing principles (K2)
- Explain the impact of context on the test process (K2)
- Describe the test activities and respective tasks within the test process (K2)
- Differentiate the work products that support the test process (K2)
- Explain the value of maintaining traceability between the test basis and test work products (K2)
- Identify the psychological factors that influence the success of testing (K1)
- Explain the difference between the mindset required for test activities and the mindset required for development activities (K2)

Synopsis

This module presents the foundational concepts of software testing, why it is needed and how it adds value to software projects, businesses, organizations and customers.

Lesson 1 – What is Testing?

OBJECTIVES

- Identify typical objectives of testing. (K1)
- Differentiate testing from debugging (K2)



3

RICECONSULTING

Lesson Objectives

By the end of this module, you should be able to achieve these objectives.

THE SOFTWARE RISK

- **Software systems are an integral part of life, from business applications (e.g., banking) to consumer products (e.g., cars).**
- **Most people have had an experience with software that did not work as expected.**



4

 RICECONSULTING

The Software Risk

Nearly everything we do today is driven by or affected by computers. Computers, in turn, are driven by software that contains the instructions of what to do and when to do it.

Software has become a part of almost everything we do, even controlling everyday things such as televisions, appliances and cars.

Software is written by people, which means there will be mistakes. When you consider all the industries and environments in which software is used, it is easy to see how dependent we are on software that works.

The problem is that software can and does fail, just when we need it the most.

When computer systems fail, all sorts of problems can result. In some cases, money can be lost, never to be recovered. Time can be wasted by having to do things over, such as the document that was lost when a word processing program unexpectedly stopped working. Businesses can lose customers due to errors when the customers lose confidence in the business to handle their money or information correctly.

Businesses may also fail to take advantage of opportunities because their existing business systems are so fragile and rigid that changes can't be made fast enough. In the worst case scenarios, lives can be lost when safety-critical systems fail. These types of losses are seen when planes fly into mountains, medical devices malfunction and missile systems fail.¹

¹ See the book "Fatal Defect" by Ivars Peterson for case studies of how computer failures caused loss of life.




Debugging - The process of finding, analyzing and removing the causes of failures in software.

Error - A human action that produces an incorrect result.

Failure - An event in which a component or system does not perform a required function within specified limits.

THE IMPACT OF SOFTWARE FAILURE

- **Money, reputation and even lives can be at stake.**
- **Consider the following examples:**
 - The Therac-25 radiation machine that maimed and killed people.
 - The patriot missile system failure to intercept a scud missile in Gulf War I
 - The NASDAQ performance failure during the Facebook IPO



5

RICE CONSULTING

The Impact of Software Failure

As we said earlier, in today's computing systems, money, reputation and even lives can be at stake. There have been many notable examples of how computer failures have impacted people in negative ways. A software defect in the Therac-25 radiation machine maimed 1 person and killed 4 others between 1985 and 1987.

http://courses.cs.vt.edu/professionalism/Therac_25/Therac_1.html

The patriot missile system's failure to intercept a scud missile in Gulf War I (1991) resulted in the deaths of 28 Americans.

<http://fas.org/spp/starwars/gao/im92026.htm>

A more recent major software failure occurred when the Facebook Initial Public Offering (IPO) went live on the NASDAQ stock exchange on May 18 2012. A coding defect led to performance problems which caused a delay in trading. As a result, to date, NASDAQ has had to pay over \$62 million. Swiss bank UBS claimed it lost over \$357 million in the failure.

<http://www.wsj.com/articles/SB10000872396390444405804577560220392935282>

<http://mic.com/articles/45233/facebook-fb-stock-rise-how-facebook-s-ipo-cost-nasdaq-10-million#.mdBgCYKpJ>

Although banking errors are normally minor, they do happen due to new systems and maintenance on older systems. One notable banking error occurred in 1996 at the First National Bank of Chicago when an ATM software error inflated 800 customer balances by sum of 763.9 billion dollars.



Quality - The degree to which a component, system or process meets specified requirements and/or user/customer needs and expectations.

THE GOAL OF SOFTWARE TESTING

- Software testing is a way to assess the quality of the software and to reduce the risk of software failure in operation.



6

RICECONSULTING

The Goal of Software Testing

Software testing is a way to assess the quality of the software and to reduce the risk of software failure in operation.

WHAT IS TESTING?

- The process consisting of all lifecycle activities, both static and dynamic, concerned with planning, preparation and evaluation of software products and related work products to determine that they satisfy specified requirements, to demonstrate that they are fit for purpose and to detect defects.
 - ISTQB Glossary 3.2

7

RICECONSULTING

The ISTQB Definition of Software Testing

This is the ISTQB definition of software testing. It is rather wordy, but contains significant concepts. Testing is:

- A lifecycle activity
- Both static and dynamic in nature
- Involves more than just running tests – Planning, preparation and evaluation of software and related work products
- A three-purpose activity:
 - Verify if requirements have been met
 - Validate if the software/system is fit for purpose
 - Find defects

ISN'T SOFTWARE TESTING JUST RUNNING TESTS?

- A common misperception of software testing is that it only consists of running tests, i.e. executing the software.
- This is *part* of testing, *but not all* of the testing activities.



8

RICECONSULTING

Isn't Software Testing Just About Running Tests?

A common misperception of software testing is that it only consists of running tests, i.e. executing the software. This is *part* of testing, *but not all* of the testing activities. In the next slide, we'll see some of the other software testing activities.

TESTING IS A PROCESS

- Software testing is a process which includes many different activities; test execution (including checking of results) is only one of these activities.

CHECKLIST



9

RICECONSULTING

Testing Is A Process

Testing is a process, just like any other software project activity. This process includes many activities that we will examine later in this module. Performing a test and checking results is only one small part of the overall testing process picture.

TESTING ACTIVITIES

- **Software test activities exist before and after test execution, such as:**
 - Test planning
 - Analyzing
 - Designing tests
 - Implementing tests
 - Reporting test progress and results
 - Evaluating the quality of a test object (the item(s) under test)



Test activities are organized and carried out differently in different lifecycles

10

RICECONSULTING

Testing Activities

In this slide we see all the testing tasks described in the ISTQB syllabus. As noted in the slide, testing activities are organized and performed differently based on the lifecycle process in use.

DYNAMIC AND STATIC TESTING

- **Some testing does involve the execution of the component or system being tested**
 - Such testing is called **dynamic testing**.
- **Other testing does not involve the execution of the component or system being tested;**
 - Such testing is called **static testing**.
- **Testing also includes reviewing work products such as requirements, user stories, and source code.**

11

RICECONSULTING

Static & Dynamic Testing

Static Testing is the testing of a component or system at specification or implementation level without execution of that software, e.g. reviews or static code analysis.

Dynamic Testing is testing that involves the execution of the software of a component or system.



Verification -
Confirmation by examination and through provision of objective evidence that specified requirements have been fulfilled.

Validation -
Confirmation by examination and through provision of objective evidence that the requirements for a specific intended use or application have been fulfilled

VERIFICATION AND VALIDATION

- Another common misperception of testing is that it focuses entirely on **verification** of requirements, user stories, or other specifications.
- While testing does involve checking whether the system meets specified requirements, it also involves **validation**, which is checking whether the system will meet user and other stakeholder needs in its operational environment(s).
 - Regardless of what is specified in a document

12



Verification and Validation

These are two terms that have largely been misapplied and misunderstood. Both are needed for complete testing.

OBJECTIVES OF SOFTWARE TESTING

- For any given project, the objectives of testing may include:
 - To evaluate work products such as requirements, user stories, design, and code
 - To verify whether all specified requirements have been fulfilled
 - To validate whether the test object is complete and works as the users and other stakeholders expect
 - To build confidence in the level of quality of the test object
 - To prevent defects
 - To find failures and defects



13



Objectives of Software Testing

In this and the following slide, we see many of the common goals of software testing. A key thing to note here is that there are many possible reasons why we might want to test something. The important thing for you to understand is to know what are your testing purposes are in a given situation. We will learn how to understand that in this course.



Test Objective

- A reason or purpose for designing and

OBJECTIVES OF SOFTWARE TESTING (2)

- **To provide sufficient information to stakeholders** to allow them to make informed decisions, especially regarding the level of quality of the test object
- **To reduce the level of risk of inadequate software quality** (e.g., previously undetected failures occurring in operation)
- **To comply with contractual, legal, or regulatory requirements or standards**, and/or to verify the test object's compliance with such requirements or standards



14



Objectives of Software Testing (2)

In the slide we see a continuation of the test objectives listed in the previous slide.

Notable additions on this list are the objectives of providing good information to stakeholders so they can make intelligent deployment decisions. This includes understanding the levels of risk and ways to reduce that risk.

Another major purpose of testing is to verify or validate that contractual legal or regulatory requirements or standards have been met.

VARYING TEST OBJECTIVES

- **The objectives of testing can vary, depending upon:**
 - **The context of the component or system being tested**
 - **The test level**
 - **The software development lifecycle model**



15



Varying Test Objectives

As we see in this slide, test objectives can vary depending upon several factors.



Debugging - The process of finding, analyzing and removing the causes of failures in software.

EXAMPLES

- **During component testing, one objective may be to find as many failures as possible so that the underlying defects are identified and fixed early.**
 - Another objective may be to increase code coverage of the component tests.
- **During acceptance testing, one objective may be to confirm that the system works as expected and satisfies requirements.**
 - Another objective of this testing may be to give information to stakeholders about the risk of releasing the system at a given time.

16

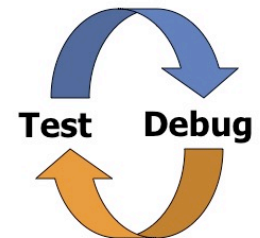


Examples

In this slide, we see some examples of adjusting the test objectives based on the level of testing.

TESTING VS. DEBUGGING

- Testing and debugging are different activities.
- Executing tests can show failures that are caused by defects in the software.
- Debugging is the development activity that finds, analyzes, and fixes such defects.



17

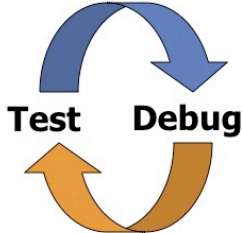


Testing vs. Debugging

There is a major difference between testing and debugging. Testing can show failures that are caused by defects. Debugging is a development activity that identifies the cause of a defect, repairs the code and checks that the defect has been fixed correctly.

TESTING VS. DEBUGGING (2)

- Subsequent confirmation testing checks whether the fixes resolved the defects.
- In some cases, testers are responsible for the initial test and the final confirmation test, while developers do the debugging and associated component testing.
 - However, in Agile development and in some other lifecycles, testers may be involved in debugging and component testing



18

RICECONSULTING

Testing vs. Debugging (2)

Subsequent confirmation testing by a tester ensures that the fix does indeed resolve the failure. The responsibility for each activity is very different. Testers test and developers debug.

In agile development projects, testers may be involved in debugging, as well as in component testing.

TESTING VS. DEBUGGING EXAMPLE

Testing

- A tester is performing a test of an accounting system.
- One of the tests indicate the calculation is not the same as contained in the expected results documented in the test case.
- The tester documents the incident and logs it as a defect report in the defect tracking tool.

Debugging

- The defect report gets assigned to the developer who wrote the code.
- Upon investigation, the developer discovers a defect in the formula used in the calculation.
- The developers corrects the formula, recompiles the software, then performs a unit test to confirm the change is correct.

- The developer's unit test passes, so the module is then sent to the tester for confirmation and regression testing.

19

RICECONSULTING

Testing vs. Debugging Example

In the above slide, we see the process of a tester discovering a failure and possible defect, as well as the investigation and debugging process, followed by confirmation and regression testing.

THE ROLE OF SOFTWARE TESTING IN A PROJECT

- **Software testing can:**
 - Find defects and identify opportunities for improvement
- **Software testing cannot:**
 - Ensure there will be no problems



20

RICECONSULTING

The Role of Testing in a Project

Always remember that testing can find defects and identify opportunities for improvement, but cannot ensure there will be no problems. This means that as a tester you cannot guarantee something is defect-free. You can only report on what you have tested and observed.

Lesson 2 – Why is Testing Necessary?

OBJECTIVES

- Give examples of why testing is necessary (K2)
- Describe the relationship between testing and quality assurance and give examples of how testing contributes to higher quality (K2)
- Distinguish between error, defect, and failure (K2)
- Distinguish between the root cause of a defect and its effects (K2)



22

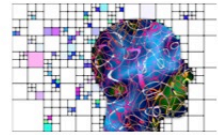
RICECONSULTING

Learning Objectives

In this slide, we see the learning objectives for this lesson.

WHY IS TESTING NEEDED?

- **Rigorous testing of components and systems, and their associated documentation, can help reduce the risk of failures occurring during operation.**
- **When defects are detected, and subsequently fixed, this contributes to the quality of the components or systems.**
 - Testing may also be required to meet contractual or legal requirements or industry-specific standards.



23

RICECONSULTING

Why is Testing Needed?

Some people believe that if the process to build something is followed correctly and consistently, there should be no need for testing. This, however, assumes that the process is perfect and that people will not make a mistake when performing it. It also assumes that the users' requirements are fully known and fully documented.

In reality, most people have flawed processes and the user requirements are incomplete, missing, or in error. On the other hand, good processes and requirements are needed for good software development. It's not good practice to rely on testing at the end of system development to find problems. The ideal way to apply testing is in a filtering approach throughout the software development life cycle.

So, testing is *one way* to find problems. Reviews can also be used to find problems throughout a project.

Of course, in order to be effective in reducing the risk of failures, the defects that have been detected through testing and reviews must be fixed. This does not always occur

TESTING'S CONTRIBUTIONS TO SUCCESS

- **Throughout the history of computing, it is common for software and systems to be deployed with defects.**
 - Defects can subsequently cause failures or otherwise not meet the stakeholders' needs.
- **However, using appropriate test techniques can reduce the frequency of such problematic deliveries when the test techniques are applied:**
 - With the appropriate level of test expertise
 - In the appropriate test levels
 - At the appropriate points in the software development lifecycle

24



Testing's Contribution to Success

One of the things we know based on many years of software projects and research is that most, if not all, software has defects in it when released. These defects can cause failures, which results in many forms of loss. Some defects can go on undetected for many years.

However, the use of the right test techniques can greatly reduce the frequency of defects seen after the release of software. This also requires that people possess an appropriate level of testing expertise at the appropriate levels of testing and at the appropriate points in the software development lifecycle.

In other words, effective testing requires:

- The right techniques
- The right skills
- The right approach
- The right timing

EXAMPLES

- **Having testers involved in requirements reviews or user story refinement could detect defects in these work products.**
 - The identification and removal of requirements defects reduces the risk of incorrect or untestable functionality being developed.
- **Having testers work closely with system designers while the system is being designed can increase each party's understanding of the design and how to test it.**
 - This increased understanding can reduce the risk of fundamental design defects and enable tests to be identified at an early stage.

25



Examples

In these examples we see ways that testers can be effective in detecting defects.

EXAMPLES (2)

- **Having testers work closely with developers while the code is under development can increase each party's understanding of the code and how to test it.**
 - This increased understanding can reduce the risk of defects within the code and the tests.
- **Having testers verify and validate the software prior to release can detect failures that might otherwise have been missed, and support the process of removing the defects that caused the failures (i.e., debugging).**
 - This increases the likelihood that the software meets stakeholder needs and satisfies requirements.

26



Examples (2)

THE ROLE OF TEST OBJECTIVES

- In addition to these examples, the achievement of defined test objectives contributes to overall software development and maintenance success.



27

RICECONSULTING

The Role of Test Objectives

In addition to the general examples just shown in the previous slides, when specifically define test objectives have been achieved, it contributes to the overall software development and maintenance success, given that the ride objectives have been defined achieved.

TESTING IS NOT QA!

- While people often use the phrase quality assurance (or just QA) to refer to testing, quality assurance and testing are not the same, but they are related.
 - A larger concept, quality management, ties them together.
 - Among other activities, quality management includes both quality assurance and quality control.
- Testing is Quality Control (QC) or the inspection of a product to find defects.

28

RICECONSULTING

Testing is Not QA!

It is helpful to understand the distinction between QA and testing. In recent years, people have erroneously used the term “QA” to mean testing. Even worse, some people use QA as a verb, such as “someone needs to QA the software”. In reality, QA is the management of quality and can include many things such as, defining processes, facilitating testing, measurement and metrics of software, and the creation and monitoring of standards.

On the other hand, testing is quality control (QC) or the inspection of a product to find defects. It’s not that QA is better or worse than QC, they are just two different activities.



Quality Assurance -


Part of quality management focused on providing confidence that quality requirements will be fulfilled.

Root Cause - A


source of a defect such that if it is removed, the occurrence of the defect type is decreased or removed.

QUALITY ASSURANCE

- Quality assurance is typically focused on adherence to proper processes, in order to provide confidence that the appropriate levels of quality will be achieved.
- When processes are carried out properly, the work products created by those processes are generally of higher quality, which contributes to defect prevention.



29




Quality Assurance


While QA does not actually assure anything, the objective of QA is to provide a level of confidence and measurement that intended software quality activities are being performed and are effective. One of the big objectives of QA is to facilitate defect prevention.

THE ROLE OF ROOT CAUSE ANALYSIS

- Root cause analysis (RCA) can help detect and remove the causes of defects for the purpose of defect prevention.
- RCA, along with the proper application of the findings of retrospective meetings to improve processes, are important for effective quality assurance.



30



The Role Of Root Cause Analysis

A root cause is the single reason that something has occurred. It is not the defect itself, but the reason for the defect. Root cause analysis is intended to study the causes of defects so preventative measures can be established.

Root cause analysis is a key way to improve processes. Project retrospective are reviews that are held after a project or sprint is complete. The purpose of the project retrospective is to discuss what went well and not so well on a particular project or sprint.

QUALITY CONTROL

- **Quality control involves various activities, including test activities, that support the achievement of appropriate levels of quality.**
- **Test activities are part of the overall software development or maintenance process.**
- **Since quality assurance is concerned with the proper execution of the entire process, quality assurance supports proper testing.**
- **Testing contributes to the achievement of quality in a variety of ways.**

31



Quality Control

Quality control involves various activities, including test activities that support the achievement of appropriate levels of quality. Test activities are part of the overall software development or maintenance process. Since quality assurance is concerned with the proper execution of the entire process, quality assurance supports proper testing. As described in earlier, testing contributes to the achievement of quality in a variety of ways.

QA AND QC EXAMPLE

- **A company embarked on a project to improve software development and testing processes.**
- **There was no QA team or processes in place at the start of the initiative.**
- **In order to know if testing (QC) processes were effective, the company realized it needed a way to measure the outcomes from testing and development across projects.**
- **So, the company established a team of 3 true QA analysts to define measurable processes for testing.**
- **After one year, the QA team had established a set of metrics which allowed the company to know which aspects of testing and development were meeting goals and also where improvements were needed.**

32



QA and QC Example

In this slide, we see the contrast and relationship between QA and QC.

WHY SOFTWARE FAILS

- **People make errors (mistakes)**
 - These produce defects (faults, bugs) in the code, in software code or in a related work product.
 - For example, a requirements elicitation error can lead to a requirements defect, which then results in a programming error that leads to a defect in the code.
- **If a defect in the code is executed, this may cause a failure, but not necessarily in all circumstances.**
 - For example, some defects require very specific inputs or preconditions to trigger a failure, which may occur rarely or never.

33



Why Software Fails

Software fails because someone makes a mistake. Although mistakes can be prevented and minimized, people are fallible and will make errors. In software, these errors can be in the understanding of what is needed, the translation of the user need into software requirement specifications, the creation of software, etc. Then, when defective software is executed in the system, a failure occurs.

It's important to realize that all failures are the result of defects, but not all defects will manifest themselves as failures in the system.

A FEW CAUSES OF SOFTWARE FAILURE

- Time pressure
- Human fallibility
- Inexperienced or insufficiently skilled project participants
- Miscommunication between project participants, including miscommunication about requirements and design
- Complexity of the code, design, architecture, the underlying problem to be solved, and/or the technologies used
- Misunderstandings about intra-system and inter-system interfaces, especially when such interactions are many
- New, unfamiliar technologies
- Environmental conditions.
 - For example, radiation, electromagnetic fields, and pollution can cause defects in firmware or influence the execution of software by changing hardware conditions.

34



A Few Causes of Software Failure

In this slide, we see a few ways that software can fail. These range from basic human fallibility, carelessness and time pressures to complexity and environmental causes. When you consider these causes, **which are only a few of the many causes**, it is no wonder that software fails.

- **Time pressure**

When the pressure to meet the deadline increases, people are more prone to take shortcuts. One of the most common shortcuts is to minimize testing.

- **Human fallibility**

People make mistakes. Even with well-defined and well-executed processes, errors can still happen.

- **Inexperienced or insufficiently skilled project participants**

- **Complexity of the code, design, architecture, the underlying problem to be solved, and/or the technologies used**

In the late 1970's, Tom McCabe wrote his paper on the observed correlation between software complexity and defects. This correlation makes sense as complex software is more difficult to test completely.

- **Misunderstandings about intra-system and inter-system interfaces, especially when such intrasystem and inter-system interactions are large in number**

The more interactions, the more chances for integration-related failures. In multiple research studies where defects were studied after the project was complete, the root cause in a very large percentage of cases (over 95%) were due to paired interactions. This is why pairwise testing is effective.²

² You can read more about this research at www.pairwise.org.

- **New, unfamiliar technologies**

Whenever embarking on projects using new technologies, there is great risk of encountering problems you have never seen or solved before.

- **Environmental causes**

In addition to failures caused due to defects in the code, failures can also be caused by environmental conditions. For example, radiation, electromagnetic fields, and pollution can cause defects in firmware or influence the execution of software by changing hardware conditions.

THE FIRST "COMPUTER BUG"

- In 1945, Grace Murray Hopper was working on the Harvard University Mark II Aiken Relay Calculator.
- On the 9th of September, 1945, when the machine was experiencing problems, a moth was found trapped between the points of Relay #70, in Panel F.
- The operators removed the moth and affixed it to the log. The entry reads: "First actual case of bug being found."



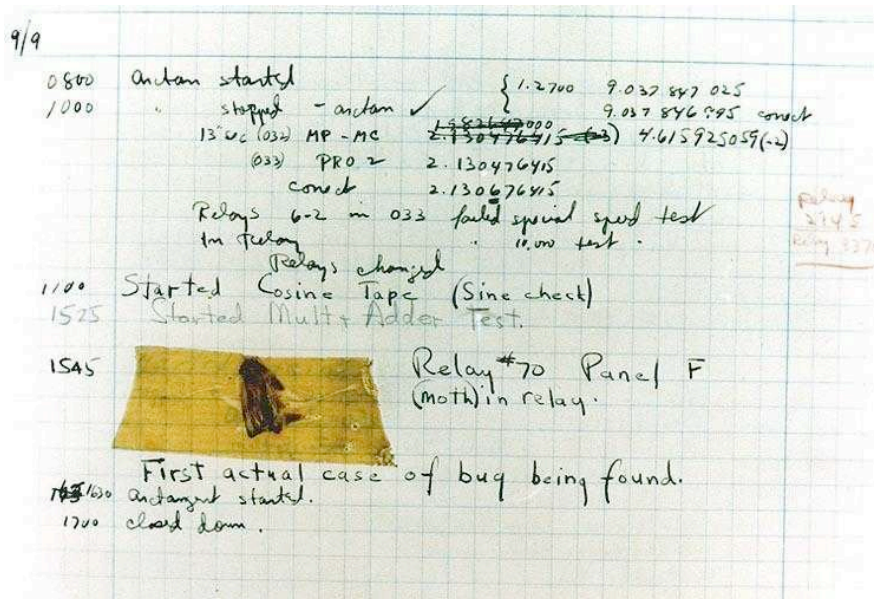
35

RICECONSULTING

The First "Computer Bug"

As an interesting side note (this is not part of the ISTQB syllabus), the first "computer bug" was found in 1945 as Grace Hopper (who later developed the COBOL language) was working on the Harvard University Mark II Aiken Relay Calculator. As it turns out, a moth was found trapped between the points of a relay. The operators removed the moth and affixed it to the log.

The entry reads: "First actual case of bug being found." The larger view of the log page can be seen below.



This is the magnified view of the actual log page of the first computer bug.

NOT ALL UNEXPECTED TEST RESULTS ARE FAILURES.

- **False positives may occur due to errors in the way tests were executed, or due to defects in the test data, the test environment, or other testware, or for other reasons.**
 - False positives are reported as defects, but aren't actually defects.
- **The inverse situation can also occur, where similar errors or defects lead to false negatives.**
 - False negatives are tests that do not detect defects that they should have detected;



37

RICECONSULTING

Not All Unexpected Test Results Are Failures

Not all unexpected test results are failures. False positives may occur due to errors in the way tests were executed, or due to defects in the test data, the test environment, or other testware, or for other reasons.

The inverse situation can also occur, where similar errors or defects lead to false negatives. False negatives are tests that do not detect defects that they should have detected; false positives are reported as defects, but aren't actually defects.

FALSE POSITIVES AND FALSE NEGATIVES

- **These can be confusing terms.**
- **Their origin is in medical testing.**
 - A positive test is bad.
 - A negative test is good.
- **Example**
 - You have a sore throat and get tested for strep throat.
 - The test results indicate you do not have strep throat.
 - The test is negative.
 - However, the problem persists and the doctor has the test performed again.
 - This time, the test results are positive. You have a strep infection.
 - The first test is an example of a false negative result.

38

 RICE CONSULTING

False Positives and False Negatives

These can be confusing terms because they sound the opposite of what we may initially expect. The terms originated in medical testing where a positive test is often bad news and a negative test result is often good news.

As an example, let's say you have a sore throat and get tested for strep throat. The test results indicate you do not have strep throat. So, the test is negative.

However, the problem persists and the doctor has the test performed again. This time, the test results are positive. You actually *do* have a strep infection.

The first test is an example of a false negative result.

FALSE POSITIVES AND FALSE NEGATIVES

- **Example**

- You have a sore throat and get tested for strep throat.
- The test results indicate you have strep throat.
 - The test result is positive and the doctor prescribes an antibiotic.
- However, the problem persists and the doctor has the test performed again.
 - This time, the test results are negative. You never had a strep infection.
 - Instead, you had a seasonal allergy
- The first test is an example of a false positive result.

39



False Positives and False Negatives

In an example of a false negative, let's say once again, you have a sore throat and get tested for strep throat. The test results indicate you have strep throat. So, the test result is positive and the doctor prescribes an antibiotic.

However, the problem persists and the doctor has the test performed again. This time, the test results are negative. So, you never actually had a strep infection. Instead, you had a seasonal allergy. The first test is an example of a false positive result.

ROOT CAUSES

- **The root causes of defects are the earliest actions or conditions that contributed to creating the defects.**
- **Defects can be analyzed to identify their root causes, so as to reduce the occurrence of similar defects in the future.**
- **By focusing on the most significant root causes, root cause analysis can lead to process improvements that prevent a significant number of future defects from being introduced.**



40



Root Causes

The root causes of defects are the earliest actions or conditions that contributed to creating the defects. Defects can be analyzed to identify their root causes, so as to reduce the occurrence of similar defects in the future. By focusing on the most significant root causes, root cause analysis can lead to process improvements that prevent a significant number of future defects from being introduced.

EXAMPLE

- A banking system calculates incorrect interest payments, due to a single line of incorrect code, which results in customer complaints.
- The defective code was written for a user story which was ambiguous, due to the product owner's misunderstanding of how to calculate interest.
- If a large percentage of defects exist in interest calculations, and these defects have their root cause in similar misunderstandings, the product owners could be trained in the topic of interest calculations to reduce such defects in the future.

41



Example

A banking system calculates incorrect interest payments, due to a single line of incorrect code, result in customer complaints.

The defective code was written for a user story which was ambiguous, due to the product owner's misunderstanding of how to calculate interest.

If a large percentage of defects exist in interest calculations, and these defects have their root cause in similar misunderstandings, the product owners could be trained in the topic of interest calculations to reduce such defects in the future.

IN THIS EXAMPLE...

- The customer complaints are **effects**.
- The incorrect interest payments are **failures**.
- The improper calculation in the code is a **defect**, and it resulted from the **original defect**, the ambiguity in the user story.
- The **root cause** of the original defect was a lack of knowledge on the part of the product owner, which resulted in the product owner making a mistake while writing the user story.

42



In This Example...

In this example, the customer complaints are effects. The incorrect interest payments are failures. The improper calculation in the code is a defect, and it resulted from the original defect, the ambiguity in the user story. The root cause of the original defect was a lack of knowledge on the part of the product owner, which resulted in the product owner making a mistake while writing the user story.

IMPORTANT TERMS

- **Defect, bug or fault:**
 - An imperfection or deficiency in a work product where it does not meet its requirements or specifications.
- **Mistake or Error:**
 - A human action that produces an incorrect result.
- **Failure:**
 - An event in which a component or system does not perform a required function within specified limits.



43

ISTQB Glossary 3.2



Important Terms

A **defect** (also known as a “**bug**” or “**fault**”) is a flaw in a component or system that can cause the component or system to fail to perform its required function. For example, a defect could be an incorrect statement or data definition. A defect, if encountered during execution, may cause a failure of the component or system.


Defects can be found in any deliverable – requirements, design, code, test cases, test automation, etc.

Also, defects may remain hidden in code for years before they are discovered by users or by testers.

Lesson 3 – General Testing Principles

OBJECTIVES

- Explain the seven testing principles. (K2)



45


RICECONSULTING

Lesson Objectives

By the end of this lesson, you should be able to explain the seven fundamental principles in software testing.

GENERAL TESTING PRINCIPLES

- A number of testing principles have been suggested over the past 50 years and offer general guidelines common for all testing.
- We will review seven major testing principles in the following slides.



46

RICECONSULTING

General Testing Principles

A number of testing principles have been suggested over the past 40 years and offer general guidelines common for all testing. We will review seven major testing principles in the following slides.

There is no single recommended way or formula to perform software testing. There could be more principles, but these seven principles are simply observations and practices that have stood the test of time as effective principles as listed in the ISTQB Foundation Level Syllabus.

PRINCIPLE 1 - TESTING SHOWS THE PRESENCE OF DEFECTS, NOT THEIR ABSENCE

- Testing can show that defects are present, but cannot prove that there are no defects.
- Testing reduces the probability of undiscovered defects remaining in the software but, even if no defects are found, testing is not a proof of correctness.



47

RICECONSULTING

Principle 1 – Testing Shows The Presence Of Defects, Not Their Absence

Testing can show that defects are present, but cannot prove that there are no defects. Testing reduces the probability of undiscovered defects remaining in the software but, even if no defects are found, it is not a proof of correctness. This means that you can't perform a few tests and predict what might be seen in the untested portions of the software or system. Even in the parts of the software that have been tested, there are degrees of testing. This means that you may test something, but still miss finding a defect.

This principle is why a wise tester will only report what they tested and what they observed, not what they predict. In other words, never bet your job that there are no more defects in something.

PRINCIPLE 2 - EXHAUSTIVE TESTING IS IMPOSSIBLE.

- Testing everything (all combinations of inputs and preconditions) is not feasible except for trivial cases.
- Rather than attempting to test exhaustively, risk analysis, test techniques, and priorities should be used to focus test efforts.



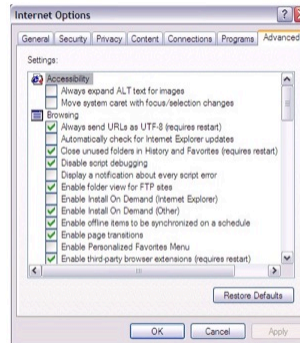
48

RICECONSULTING

Principle 2 – Exhaustive Testing Is Impossible

Testing everything (all combinations of inputs and preconditions) is not feasible except for trivial cases. Instead of exhaustive testing, we use risk and priorities to focus testing efforts. We will explore this principle in the following slides with an example.

AN EXAMPLE



The IE Tools>Advanced Options window

53 binary conditions
1 condition with 3 options
1 condition with 4 options

$$2^{53} = 9,007,199,254,740,992 \times 12$$

108,086,391,056,891,904
Possible combinations of conditions!

49

RICECONSULTING

An Example

In Internet Explorer, under the Tools menu option, there is a sub-option for “Internet Options”. There is a tab on the dialog for “Advanced” settings. Now, keep in mind that this is just ONE small (yet important) feature of IE. In the advanced settings, there are 53 binary (on/off) conditions, with one condition with 3 options and another with 4 options. If you compute the possible combinations, there are 108 quadrillion, 86 trillion, 391 billion, 56 million, 891 thousand, nine hundred and four.

THIS WOULD REQUIRE...

At one second per test execution:

$$\frac{108,086,391,056,891,904}{3,600} = 30,023,997,515,803.3 \text{ hours}$$

$$1,250,999,896,491.8 \text{ days}$$

$$3,427,396,976.69 \text{ years}$$

To test all possible combinations!

50



This Would Require...

To see the impact of this, at one second per test execution, it would take over 30 trillion hours, over 1 trillion days, and over 3.4 billion years to test all the possible combinations of options in this one dialog alone! When you consider this is just functional testing and doesn't include the testing of non-functional attributes, such as security, performance, and so forth, it makes the job of testing even more daunting. So, we see from this little example that complete testing is impossible in software with any level of complexity. Most graphical user interfaces contain enough combinations to reach the billions of possible tests.

PRINCIPLE 3 – EARLY TESTING IS GOOD

- To find defects early, both static and dynamic test activities should be started as early as possible in the software development lifecycle.
- Early testing is sometimes referred to as “shift left” .
- Testing early in the software development lifecycle helps reduce or eliminate costly changes



51



Principle 3 – Early Testing Is Good

Testing activities should start as early as possible in the software or system development lifecycle, and should be focused on defined objectives. The phrase you will often hear is “Test early and often.” Let's look at some research about why this is a key testing principle.

PRINCIPLE 4 – DEFECTS TEND TO CLUSTER

- A small number of modules usually contains most of the defects discovered during pre-release testing, or is responsible for most of the operational failures.
- Predicted defect clusters, and the actual observed defect clusters in test or operation, are an important input into a risk analysis used to focus the test effort.



52

RICECONSULTING

Principle 4 – Defects Tend To Cluster

A small number of modules contain most of the defects discovered during pre-release testing, or show the most operational failures. This is also the 80/20 rule applied to defects. That is, 20% of the code (or less) can contain 80% of the defects. What this means to testing is that if you find one defect, you should look more closely to see if there are others in the same area of the software or system.



Test Case - A set of preconditions, inputs, actions (where applicable), expected results and postconditions, developed based on test conditions.

PRINCIPLE 5 – BEWARE OF THE PESTICIDE PARADOX

- **If the same tests are repeated over and over again, eventually these tests no longer find any new defects.**
 - Tests are no longer effective at finding defects, just as pesticides are no longer effective at killing insects after a while.
- **To detect new defects, existing tests and test data may need changing, and new tests may need to be written.**
 - In some cases, such as automated regression testing, the pesticide paradox has a beneficial outcome, which is the relatively low number of regression defects.



53

RICECONSULTING



Principle 5 – Beware of The Pesticide Paradox

If the same tests are repeated over and over again, eventually the same set of test cases will no longer find any new bugs. To overcome this, the test cases need to be regularly reviewed and revised, and new and different tests need to be written to exercise different parts of the software or system to potentially find more defects.

This was first written about by Boris Beizer in the 1980's in his book, "Software Testing Techniques." The idea is based on something that happens when eradicating insects with pesticides. Over a period of time, a particular pesticide will become ineffective because the weaker bugs are killed, but the stronger ones survive and breed offspring that are resilient to the pesticide. This also happens with antibiotics and germs.

Although software bugs are not living organisms, the paradox remains the same. Tests that used to be effective in finding software bugs become less effective as more defects are found. This is why testers should always reevaluate their tests to make sure that new conditions are being tested or that old conditions are being tested in different ways.

Further Resources

The Art of Software Testing, 3rd Ed by Glenford Myers

Lessons Learned in Software Testing by Kaner, Bach and Pettichord

Perfect Software and Other Illusions about Testing by Gerald Weinberg

PRINCIPLE 7 – ABSENCE-OF-ERRORS IS A FALLACY

- It is a fallacy (i.e., a mistaken belief) to expect that just finding and fixing a large number of defects will ensure the success of a system.
- For example, thoroughly testing all specified requirements and fixing all defects found could still produce a system that is difficult to use, that does not fulfill the users' needs and expectations, or that is inferior compared to other competing systems



55

RICECONSULTING

Principle 7 – The Absence-Of-Errors Fallacy

Finding and fixing defects does not help if the system built is unusable and does not fulfill the users' needs and expectations.

Validation is how it is shown that the system meets user needs. The reason that validation is needed is because it has been learned over the years that tests may pass as defined on paper. However, in the real world the software may fail because the real world condition was never considered as a test condition because it was never defined as a written requirement for the software.

So the absence-of-errors fallacy means that just because the software passes all the tests that are performed does not necessarily mean it will meet the users needs.

Discussion Question

A question to ask or discuss at this point, is “How have you seen these testing principles in action in your projects?”

For example, have you ever found a defect during testing and then found another defect very similar to it in a similar location in the software?

Remember, that this is an area of the certification exam that you will be expected to know and understand.