Software Quality Assurance

Integrating Testing, Security, and Audit

Abu Sayed Mahfuz

CRC Press
Taylor & Francis Group
Boca Raton London New York

CRC Press is an imprint of the
Taylor & Francis Group, an **informa** business
AN AUERBACH BOOK

CRC Press
Taylor & Francis Group
6000 Broken Sound Parkway NW, Suite 300
Boca Raton, FL 33487-2742

Printed on acid-free paper
Version Date: 20151124

International Standard Book Number-13: 978-1-4987-3553-7 (Hardback)

Library of Congress Cataloging-in-Publication Data

Names: Mahfuz, Abu Sayed, author.
Title: Software quality assurance : integrating testing, security, and audit / author, Abu Sayed Mahfuz.
Description: Boca Raton : Taylor & Francis, 2015. | Includes bibliographical references and index.
Identifiers: LCCN 2015045370 | ISBN 9781498735537 (alk. paper)
Subjects: LCSH: Computer software--Quality control. | Computer software--Testing.
Classification: LCC QA76.76.Q35 M34 2015 | DDC 005.3028/7--dc23
LC record available at http://lccn.loc.gov/2015045370

Contents

7
Defect Management

Introduction

This chapter, as the name implies, deals with the conceptual aspects of defect management. There are three parts in this chapter. Part 1 discusses the basic concepts of a defect and why a defect happens. Part 2 introduces the practical methodologies of how to manage the defects. In this section, some sample documents and templates are provided to manage the defect properly. Part 3 discusses and analyzes the root causes of defects and provides recommendations of how to prevent defects in the future.

Part 1: Definition and Analysis

Definitions

Defect A defect in simple terms is a variance from expectation. Another definition is that a defect is a condition in a process/product which does not meet a documented requirement. In other words, a defect is an error in a process or product's behavior that causes it to malfunction or to produce incorrect or unexpected results.

The root cause of a defect may originate from different sources such as code, requirements, design, environment, build/compilation, test case, and data.

Defect in Hardware In IEEE 610, defect or fault is defined as "A defect in a hardware device or component; for example, a short circuit or broken wire."

Defect in Software "An incorrect step, process, or data definition in a computer program."*

This definition is used primarily by the fault tolerance discipline. In common usage, the terms "error" and "bug" are used to express this meaning.

Definition of an Error

In IEEE 610, the error is defined as

- "The difference between a computed, observed, or measured value or condition and the true, specified, or theoretically correct value or condition."
    - For example, a difference of 30 m between a computed result and the correct result.
- "An incorrect step, process, or data definition."
    - For example, an incorrect instruction in a computer program.
- "An incorrect result."
    - For example, a computed result of 12 when the correct result is 10.
- "A human action that produces an incorrect result."
    - For example, an incorrect action on the part of a programmer or operator.†

Defect Repository Defect repository is the defect management tool/repository used to track defects and all defects associated with the application under test. There are many tools available. However, many companies typically use HP Quality Center or IBM Clear Quest for defect repository.

What Causes Defects in Software

Certainly, it is very important and necessary to understand why an error happens. Honestly speaking, that is the point. When the cause

* IEEE Std 610.12-1990 (Revision and redesignation of IEEE Std7SZ.1983) *IEEE Standard Glossary of Software Engineering Terminology*, p. 32.

† Ibid., p. 31.

could be identified, almost half of the problem is resolved. In this arena, it is called defect root cause. In this chapter, we have dedicated one important part on root cause analysis (RCA), and we will try to evaluate them.

When the software code has been built, it is executed and then any defects may cause the system to fail to do what it should do (or do something it should not), causing a failure. Interestingly and alarmingly, sometimes a defect may not be obvious even though it exists; in programming language, it is called a logic error.

In computer programming, a logic error is a bug in a program that causes it to operate incorrectly, but not to terminate abnormally (or crash). A logic error produces unintended or undesired output or other behavior, although it may not immediately be recognized as such.

Logic errors may occur in both compiled and interpreted languages. Unlike a program with a syntax error, a program with a logic error is a valid program in the language, even though it does not behave as intended. The only clue to the existence of logic errors is the production of wrong solutions.

This example function in C is to calculate the average of two numbers that contains a logic error. It is missing brackets in the calculation so it compiles and runs but does not give the right answer due to operator precedence (division is evaluated before addition).

```
int average (int a, int b)
{
    return a + b / 2;     /* should be (a + b) / 2*/
}
```

In simple math,

(2 + 5) × 2 and 2 + 5 × 2 is not the same.

In (2 + 5) × 2 the result is 14; on the other hand, 2 + 5 × 2 = 12 because here you have to do the multiplication first then the addition. There is a rule called PEMDAS which represents as

P	= Parenthesis
E	= Exponents
M	= Multiplication
D	= Division
A	= Addition
S	= Subtraction

In these circumstances, if the developer writes the wrong code such as forgetting to put (2 + 5) in parenthesis, then the result will be wrong even though for the tester it may look like the correct result.

Detecting a Defect Early

It is indeed better to find the defect as early as possible. In software development, if there is a mistake in requirement and you found it in production that could lead to a big mess.

To fix this defect, no matter how much it may cost, you may not find all people you need—developer, tester, designer—everyone may not be available then, and it is not a simple thing. For example, it is easier to build a new building than repair an old building. It could be a total disaster.

What Is the Cost of Defects Not Detected Early?

In addition, considering the impact of failures arising from defects, which we have not found, we need to consider the impact once we find those defects. The cost of finding and fixing defects rises considerably across the life cycle; think of the old English proverb "a stitch in time saves nine." This means that if you mend a tear in your sleeve now while it is small, it is easy to mend; but if you leave it, it will get worse and need more stitches to mend it.

When a defect exists in the requirement specification and is not detected until acceptance testing or until production, then it will be much more expensive to fix (Figure 7.1).

It is quite often the case that defects detected at a very late stage are not corrected because the cost of doing so is too expensive. Also, if the software is delivered and meets an agreed specification, if the specification was wrong, then the software will not be accepted. The project team may have delivered exactly what they were asked to deliver, but it is not what the users wanted. In some cases, where the defect is too serious, the system may have to be completely reinstalled.

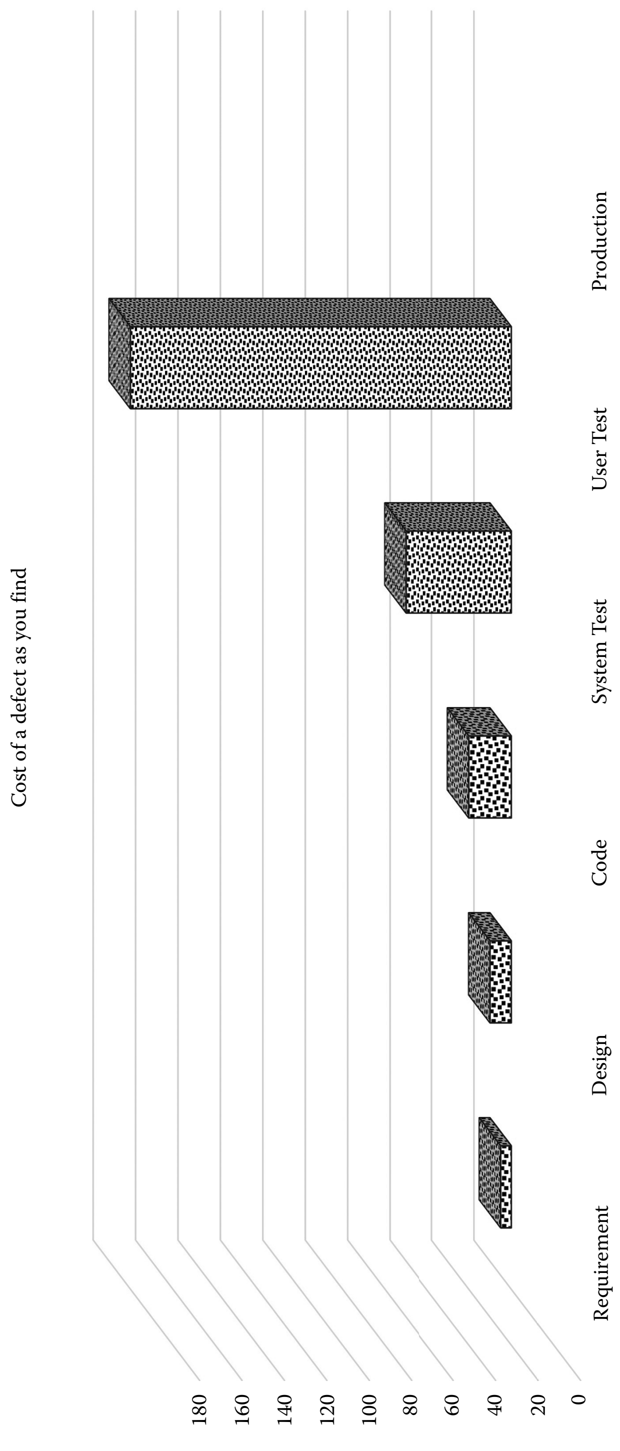

Figure 7.1 The cost of a defect is much higher in production than in requirement.

Defect Life Cycle Steps

The IEEE 1044 defect life cycle consists of the following four steps (Figure 7.2):

Step 1: Recognition or Identification Recognition occurs when we observe an anomaly, that observation being an incident, which is a potential defect. This can occur in any phase of the software life cycle.

Step 2: Investigation After recognition, the investigation of the incident occurs. Investigation can reveal related issues. Investigation can propose solutions. One solution is to conclude that the incident does not arise from an actual defect; for example, it might be a problem in the test data.

Step 3: Action The results of the investigation trigger the action step. We might decide to resolve the defect. We might want to take action indicated to prevent future similar defects. If the defect is resolved,

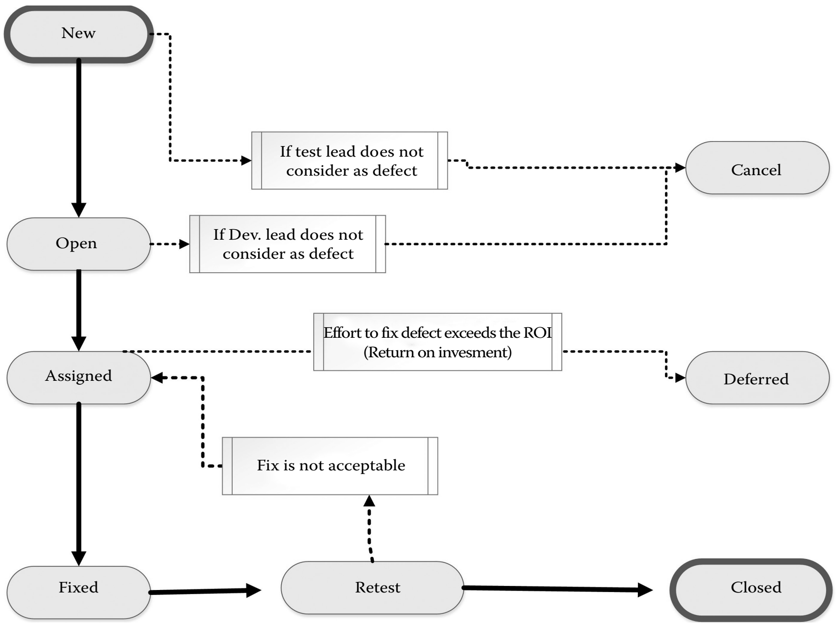
New
If test lead does not consider as defect
Cancel
Open
If Dev. lead does not consider as defect
Effort to fix defect exceeds the ROI (Return on invesment)
Assigned
Deferred
Fix is not acceptable
Fixed
Retest
Closed

Figure 7.2 Defect life cycle from new to closed.

regression testing and confirmation testing must occur. Any tests that were blocked by the defect can now progress.

Step 4: Disposition With action concluded, the incident moves to the disposition step. Here, we are principally interested in capturing further information and moving the incident into a terminal state.

Objectives of Testing

Reduce the Risk of Failure Most of the complex software systems contain faults, which cause the system to fail from time to time. This concept of "failing from time to time" gives rise to the notion of *failure rate*. As faults are discovered and fixed while performing more and more tests, the failure rate of a system generally decreases. Thus, a higher level objective of performing tests is to bring down the risk of failing to an acceptable level.

Reduce the Cost of Testing The different types of costs associated with a test process include the cost of designing, maintaining, and executing test cases; the cost of analyzing the result of executing each test case; the cost of documenting the test cases; and the cost of actually executing the system and documenting it.

Therefore, the fewer test cases designed, then the cost of testing is reduced. However, producing a small number of arbitrary test cases is not a good way of saving money. The highest level objective of performing tests is to produce low-risk software with fewer test cases. This idea leads us to the concept of *effectiveness of test cases*. Therefore, the test engineers must judiciously select fewer, more effective test cases.

Analyze Root Causes

According to Capability Maturity Model Integration (CMMI), the objective of defect RCA is to determine causes of defects.

Root causes of defects and other problems are systematically determined.

Address Causes of Defects Root causes of defects and other problems are systematically addressed to prevent their future occurrence.

Institutionalize a Defined Process A root cause is a source of a defect; if it is removed, the defect is decreased or removed.

Determine which defects and other problems will be analyzed further.

When determining which defects to analyze further, consider the impact of the defects, the frequency of occurrence, the similarity between defects, the cost of analysis, the time and resources needed, safety considerations, and so on.

Perform causal analysis of selected defects and other problems and propose actions to address them.

The purpose of RCA is to develop solutions to the identified problems by analyzing the relevant data and producing action proposals for implementation.

Conduct causal analysis with the people who are responsible for performing the task.

Causal analysis is performed with those people who have an understanding of the selected defect or problem under study, typically in meetings.

An action proposal usually documents the following:

Originator of the action proposal
Description of the problem
Description of the defect cause
Defect cause category
Phase when the problem was introduced
Phase when the defect was identified
Description of the action proposal
Action proposal category

Projects operating according to a well-defined process will systematically analyze the operation where problems still occur and implement process changes to eliminate root causes of selected problems.

Implement the Action Proposals

Implement the selected action proposals that were developed in causal analysis.

Action proposals describe the tasks necessary to remove the root causes of the analyzed defects or problems and avoid their recurrence.

Only changes that prove to be of value should be considered for broad implementation.

Part 2: Process and Methodology

Defect Management Process

There are several high-level steps to be taken in a typical defect management process. The following items are highly recommended, which are also supported by IEEE standards.

Identifying

The first thing that needs to be done is to identify the defect: what is it and how did this happen? The first person who identifies the defect should submit it as defect to his or her lead and the team lead should evaluate, verify, and identify it as a defect, and then it remains an open defect.

Categorizing

When a defect is reported and verified by the test team, it remains as open, then it should be assigned to someone, usually to a related developer. Once the defect is categorized, the defect moves on in the process to the next step that is prioritization.

Prioritizing

Prioritization is typically based on a combination of the severity of impact on the user, relative effort to fix, along with a comparison against other open defects. The priority should be determined with representation from the management, the customer, and the project team.

Assigning

Once a defect has been prioritized, it is then assigned to a developer or other technician to fix.

Resolving

The developer fixes (resolves) the defect and follows the organization's process to move the fix to the environment where the defect was originally identified.

Verifying

Depending on the environment where the defect was found and the fix was applied, the software testing team or customer typically verifies that the fix has actually resolved the defect.

Closing

Once a defect has been resolved and verified, the defect is marked as closed.

Management Reporting

Management reports are provided to appropriate individuals at regular intervals as defined reporting requirements. In addition, on-demand reports are provided on an as-needed basis.

Roles and Responsibilities in Software Development Life Cycle

Business Owner

The business owner requests funding, sets business requirements, and works with the technology owner to make strategic decisions.

Stakeholders

Stakeholders include anyone who will be impacted by a project, including security, risk, compliance, and governance organizations.

Stakeholders should actively work with the analysts, testers, and developers to ensure that the defects have been logged and addressed in a timely manner, participate in defect review meetings, and provide input into the final defect disposition.

Analyst

The analyst's role is responsible for reviewing any defects that impact business and system operations. The analyst should participate or represent someone in defect review meetings to ensure that proper severity and priority have been assigned to the defects. The analyst should work with the testing and development team to confirm that the defects have been properly fixed and retested.

Developer

A developer is responsible for researching and remediating any assigned defects that have been opened by the testers or any other stakeholders. Developers should work with the testers and analysts to provide additional defect information, research the defect, and provide a fix to prevent the defect from recurring. Developers must participate in defect review meetings and provide updates to the defect fixes that are pending disposition as well as discuss any temporary workarounds that apply until a permanent fix is identified and implemented.

Tester

A tester is a project team member performing testing activities such as system testing or user testing. Testers are responsible for testing the application, registering and tracking all testing defects, and documenting any issues that will need to be escalated or reviewed with the management. Testers should work with the business and development team to determine priorities, severities, and remediation dates. Testers must participate in defect review meetings to ensure that all defects are tracked and are appropriate.

Conflict Resolution and Escalations during Defect

If there is any dispute or disagreement regarding any defect or about the interpretation of any terminology, and if the dispute cannot be resolved within the business unit, the business owner shall attempt to resolve the dispute.

Defect Management Methodology

Identifier:
Effective Date: mm/dd/yyyy
Version: 0.00

Document Change Control

VERSION CHANGE DATE	VERSION #	WHAT KIND OF CHANGE/ REVISION	WHERE CHANGED HAPPENED (SECTION/PAGE)	REVISED BY NAME AND TITLE	APPROVED BY NAME AND TITLE

Documentation

PROCEDURE NAME	DEFECT MANAGEMENT PROCEDURE	COMMENT
Version number:	1.00	
Procedure identifier:		
Superseded procedure(s):	N/A	
Date approved:		
Effective date:		
Procedure author(s):		
Procedure owner:		
Procedure approver:		
Procedure repository:		
Supporting documentation:	Defect management standard Software development life cycle (SDLC) Standard End user computing (EUC) standard Security vulnerability remediation standard Infrastructure hardware change (IHC) procedure Incident management standard Incident management procedure Technology incident management standard Technology incident management procedure	

Statement of Purpose

The primary goal of this procedure is to provide clear definitions and a list of values for all software defect attributes. This procedure will ensure that all defect repositories will use a consistent defect reporting and management process.

Risks

It is important that the company or team exhibit suitable and effective controls managing defects, ensuring their timely resolution based on their severity, and update their resolution progress to the stakeholders as it is critical for the company's finance and other key business processes.

Defect Steps

STEP #	STATUS	DESCRIPTION	PRIMARY PERFORMER	OUTPUT/ EVIDENCE	INPUT
1. SUBMITTING THE DEFECT					
1	New	The submitter identifies and records in a defect repository.	Test team Business team	Defect record is "submitted" in the defect repository.	
2. RESOLVING THE DEFECT (PENDING)					
2	Open pending resolution	Acknowledge the submitted defect. The defect is assigned to the development team.	Development team/ business team	Defect record is moved to a "pending resolution" state in the defect repository.	
3. RESOLVING THE DEFECT (RESOLVED)					
3	Fix and resolved	Resolve the acknowledged defect. The defect is sent to the submitter for re-test.	Development team	Defect record is moved to "resolved" state in the defect repository.	

(*Continued*)

STEP #	STATUS	DESCRIPTION	PRIMARY PERFORMER	OUTPUT/ EVIDENCE	INPUT
4. REOPENING THE DEFECT					
4	Reopen pending resolution	The submitter retests the resolved defect and the defect still exists. Reopen the resolved defect. The defect is reassigned to the development team for further analysis or resolution.	Test team Business team	Defect record is moved to a "pending resolution" state in the defect repository.	
5. CLOSING THE DEFECT					
5	Closed	The submitter retests the resolved defect and the defect does not exist. Close the resolved defect.	Test team Business team.	Defect record is moved to "closed" state in the defect repository.	
6. DEFERRING THE DEFECT					
6	Deferred	Business team determined to defer the acknowledged defect.	Business team	Defect record is moved to "deferred" state in the defect repository.	
7. CANCELING THE DEFECT					
7	Cancelled	Business or development team cancels the acknowledged or deferred defect. Submitter concurs.	Test team Business team	Defect record is moved to "cancelled" state in the defect repository.	

Defect States

The mandatory states of a defect are
- Submitted/New
- Open
- Resolved
- Closed
- Canceled
- Deferred

ATTRIBUTES	DESCRIPTION	TYPE	OPTIONAL (O) REQUIRED (R) CONDITIONALLY REQUIRED (CR)	BUSINESS RULES
Defect ID	Defect name of ID	System list	R	
Project ID	Project name or ID	System list	R	Name or ID of the project. Most recent project name or ID must be used if a defect is found in production.
Application	System in which defect was identified	List	R	Unique application identifier. Should match to the CMDB and asset ID databases.
Functional area	Module/ subsystem/ component in which defect was identified	System list	R	List of functional areas, modules, or components. The defect is associated with an application.
Headline	One line summary of the defect	Text	R	
Description	Detailed description of problem that includes steps to reproduce the defect, actual results, and expected results	Text	R	
SDLC phase found	Phase where the defect was detected	List	R	Analysis (optional), design (optional), construction, system test, user test, implementation and postimplementation.
Found in environment	Environment ¡n which defect was found	List	CR	Development, test, acceptance, production, integration and contingency. Not required if root cause is requirement or design.
Found in release number	Release number in which the defect is found	System list	R	

(*Continued*)

ATTRIBUTES	DESCRIPTION	TYPE	OPTIONAL (O) REQUIRED (R) CONDITIONALLY REQUIRED (CR)	BUSINESS RULES
Closed in release number	Release number for which the defect is closed	System list	CR	Conditionally required if the state is closed
Remedy incident ID	Problem ticket number for defects found in production	Text	CR	Conditionally required if the SDLC phase found is implementation or postimplementation
Test case ID	Associated test case ID	Text	CR	Not required if root cause is requirement or design
Functionality type	Lists the type of functionality that introduced the defect	List	R	New functionality, existing functionality
Severity	Impacts to application functionalities, business processes, or interfaces causing minor to critical disruption to application usage	List	R	Severity of the defect must be set in consensus with stakeholders to one of the following: Critical High Medium Low

Defect Attributes

When a defect is discovered, the following minimum set of defect information must be reported in the defect repository.

ATTRIBUTES	DESCRIPTION	TYPE	OPTIONAL (O) REQUIRED (R) CONDITIONALLY REQUIRED (CR)	BUSINESS RULES
Priority	Prioritizing defect based on how fast the defect should be fixed	List	R	Priorities will be set by the submitter of the defect to one of the following: Critical High Medium Low

(*Continued*)

ATTRIBUTES	DESCRIPTION	TYPE	OPTIONAL (O) REQUIRED (R) CONDITIONALLY REQUIRED (CR)	BUSINESS RULES
Root cause	Analysis of what caused the defect	List	R	
Root cause reason	Provide the reason for the root cause of the issue.	List	CR	Required if root cause is: Requirements Incomplete/missing Unclear Inconsistent Incorrect Not traceable Not testable
Resolution notes	Details regarding the resolution of defect	Text	CR	Conditionally required if the state is not submitted
State	Provides a current state of the defect's current flow while going through defect resolution process	System action	R	This is the defect state at any point in time. The values in this field are auto populated
Deferral Business Impact	Business impact description if defect is deferred as well as work around if applicable	Text	CR	Required if the defect is deferred
Submitter	Name of person that created the defect	System generated	R	
State updated By	Name of person that last changed the state of the defect	System list or system generated	CR	Name of the person that last change the state. Required if the state of the defect is deferred, closed, or canceled.
Update date	Date on which defect state	System	R	Defect repository will maintain the audit trail for the defect
Workaround	Lists the work around for deferred defects	Text	CR	Required if the state of the defect is deferred
System test	Identifies whether the defect was leaked from system test	List	CR	Required if the SDLC phase found is user test of higher Yes No

Defect Priorities

The defect priorities and definition may differ from one testing stage to another; sometimes in some projects, it may also differ from person to person. The basic definitions are provided below. The expected resolution timeframe for the defects depends on their priority.

Security defects have a prescribed timeframe for remediation that are spelled out in the security, vulnerability, and remediation standard.

PRIORITY DESCRIPTION	PRIORITY DEFINITIONS FOR DEFECTS IN NONPRODUCTION	PRIORITY DEFINITIONS FOR PRODUCTION DEFECTS
Critical	Immediate attention—critical may also mean that it might be blocking some other activities. The critical defect should be resolved immediately.	Immediate attention—must receive highest development priority and should be resolved immediately.
High	Should be reported immediately to the development team. A response or action plan must be provided within 2 working days since the defect causes more than one of the functional areas to be untestable.	Should be reported immediately to the development team. A response or action plan must be provided within 2 working days.
Medium	A response or action plan should be reported within 5 working days.	A response or action plan should be reported within 5 working days. This defect should be resolved in the next release.
Low	Fix dates are subject to negotiation. An action plan before the next release.	Fix dates are subject to negotiation. An action plan before the next release.

Defect Severities

The defect severity definition may also differ from one testing stage to another and also may differ among stakeholders as human perception may be different.

SEVERITY DESCRIPTION	SEVERITY DEFINITIONS FOR SYSTEM, USER, AND PRODUCTION	SEVERITY DEFINITIONS FOR REQUIREMENT DEFECTS
Critical	Critically severe defect causes severe business disruption, financial or reputational impact, and no workaround exists. The customer is unable to use the product, resulting in a critical impact to their operation. This defect must be resolved before exiting current phase or releasing to production.	The reason for the requirement defect could be considered such as Incomplete/missing Inconsistent Incorrect

(*Continued*)

SEVERITY DESCRIPTION	SEVERITY DEFINITIONS FOR SYSTEM, USER, AND PRODUCTION	SEVERITY DEFINITIONS FOR REQUIREMENT DEFECTS
High	Significant business disruption but a workaround exists. The customer is able to use the product but is severely restricted. This defect should be resolved before exiting current phase or releasing to production.	Content has a major inaccuracy or is missing important detail. The reason for the requirement defect could be considered such as Incomplete/missing Incorrect Unclear Inconsistent Not traceable Not testable
Medium	Minor business disruption but has a workaround, minor usability issues. This defect should be resolved before exiting current phase or releasing to production.	Content is correct but has a moderate flaw that needs amendment; for instance, because it is unclear, imprecise, or not concise. The reason for the requirement defect could be considered such as Unclear Not traceable Not testable
Low	The defect may be cosmetic in nature or a usability annoyance, such as warning messages, misspelled words, etc.	Formatting or organizational observation or a grammatical or spelling error not affecting the meaning. The reason for the requirement defect is usually unclear

Part 3: Root Cause Analysis

Definition

A root cause is an originating cause of either a condition or a causal chain that leads to an outcome or result.

In software development, we can see how defects may arise and are caused by any field.

Root cause of a defect identifies the process or source that introduced the defect.

Root Cause Fields

Standard acceptable values for the root cause field are listed in the following sections:

Requirements This field is required if the root cause of the defect is requirements and should list the actual issue of the stakeholder requirement document that introduced the defect.

Defect Cause in Requirement The possible cause of defect in requirement could be

Incomplete/Missing Necessary functionality is omitted from the requirements set. The defect should specify what needs to be documented to address the gap.

Unclear: A requirement is not simple, specific, clear, and unambiguous.

Inconsistent A requirement is in conflict with one or more other requirements or other requirements make it redundant.

Incorrect A requirement does not reflect the needs of one or more project stakeholders.

Not Traceable A requirement cannot be traced to project scope, that is, it cannot be established as being within the approved scope of the project.

Not Testable A requirement does not specify observable functionality and so cannot be validated by testing.

Implementation Dependent A requirement does not describe desired system functionality independent of the technology and design that will be used to achieve it.

Design This root cause should be selected if the solution specifications or detailed design are missing, inconsistent with requirements, or otherwise incorrect.

Code This root cause should be selected if the application failed to produce expected result or the functionality is missing, not consistent with stakeholder requirements, solution specifications, standards, or otherwise incorrect.

Environment This root cause should be selected if the application failed to produce expected results due to incorrect environment, infrastructure, or application configuration set up. Examples of environment issues are errors in software compilation/build, incorrect application configuration settings, application processes not initialized, application password is expired, third-party packages are missing, dependent systems are not available, and so on.

Test This root cause should be selected if a defect was reported incorrectly because of inconsistent test case results with stakeholder requirements or solution specifications, premature test case execution, or the test case was not executed in the appropriate environment.

Data This root cause should be selected if the product failed to produce expected results due to the improper setup of test data in the pertinent databases or input files.

Analysis RCA is an effective technique to investigate the origin for defect occurrence. This analysis helps to prevent reoccurrences of defect in future.

The Most Common Root Cause Classifications

Despite the existence of various rationales, RCA techniques enable to classify the most common root causes and percentage of their contributions toward various defect patterns. They are communication (25%–30%), education (20%–25%), oversight (30%–40%), transcription (20%–25%), and miscellaneous (5%–10%). From the defect distribution and defect pattern analysis, it is evident that trivial defects contribute more toward defect injection (Table 7.1).

TABLE 7.1 A Sample Report of Defects in Different Stages and Severity Level Report, and % of Pass and Fail

PLANNING PHASE	
METRICS DATA ELEMENT	VALUE
Planned progression coverage	100%
Prerelease overall regression capability	90%
Prerelease overall regression automation	70%
Planned regression coverage	100%
Planned regression coverage—automated	40%
Expected initial pass rate	90%
EXECUTION PHASE	
METRICS DATA ELEMENT	VALUE
Actual progression coverage	100%
Progression coverage—automated	50%
Actual regression coverage	80%
Postrelease overall regression capability	
Postrelease overall regression automation	
Actual initial pass rate	80%
Final pass rate	100%
System test defects—critical/high	5
System test defects—medium/low	12
System test defects—deferred	2
PRE-PROD PHASE	
METRICS DATA ELEMENT	VALUE
UAT defects—critical/high	2
UAT defects—medium/low	3
UAT defects—deferred	1
System test leakage—critical/high	0
System test leakage—medium/low	1

In Table 7.1, in the planning phase:

Progression coverage was planned to cover 100% and actually 100% was covered.

Prerelease overall progression was planned to automate 70%, but the actual progression automation was covered 50%.

Regression coverage was planned to cover 60%, but actually team was able to cover 80%.

Initial pass rate was expected (as usually some tests fail) to reach 90% (line 9); however, the actual initial pass rate was 80% (line 19), which is less than expected. This also means more test cases failed than expected, but the good news is that the final pass rate is 100%, which means the development team was able to resolve and fix the defects.

There are 11 defects found by the system test team, where the severity level of 3 of them were critical/high (line 21) and 8 of them were medium/low (line 22); on the other hand, UAT or the user acceptance test team, found 5 defects in total, 4 which were already found by the test team, 1 defect that was found by UAT but the system test failed to find it defined and system test leakage.

So altogether, there were 12 defects; among these, 9 defects were resolved and closed, and 3 could not be resolved at this time. The team including stakeholders decided to work around for now and possibly resolve it in future release. These 3 unresolved defects are called deferred (Figures 7.3 and 7.4).

DEFECT ID	STATE	SDLC PHASE FOUR	ROOT CAUSE	DEFECT PHASE AGE
1001	Closed	System Test	Requirement	3
1002	Closed	Design	Code	–1
1003	Closed	System Test	Design	2
1004	Closed	System Test	Requirement	3
1005	Closed	System Test	Code	1
1006	Deferred	System Test	Code	1
1007	Deferred	User Test	Code	2
1008	Closed	Design	Code	–1
1009	Closed	Design	Requirement	1
1010	Closed	System Test	Code	1
1011	Closed	User Test	Design	3
1012	Deferred	User Test	Code	4

Defect Prevention

Awareness of defect injecting methods and processes enables defect prevention (DP). It is the most significant activity in software development. It identifies defects along with their root causes and prevents their recurrences in the future.

Benefits of Defect Prevention Prevention is better than a cure; it applies to defects as well. It is indeed better to prevent a defect

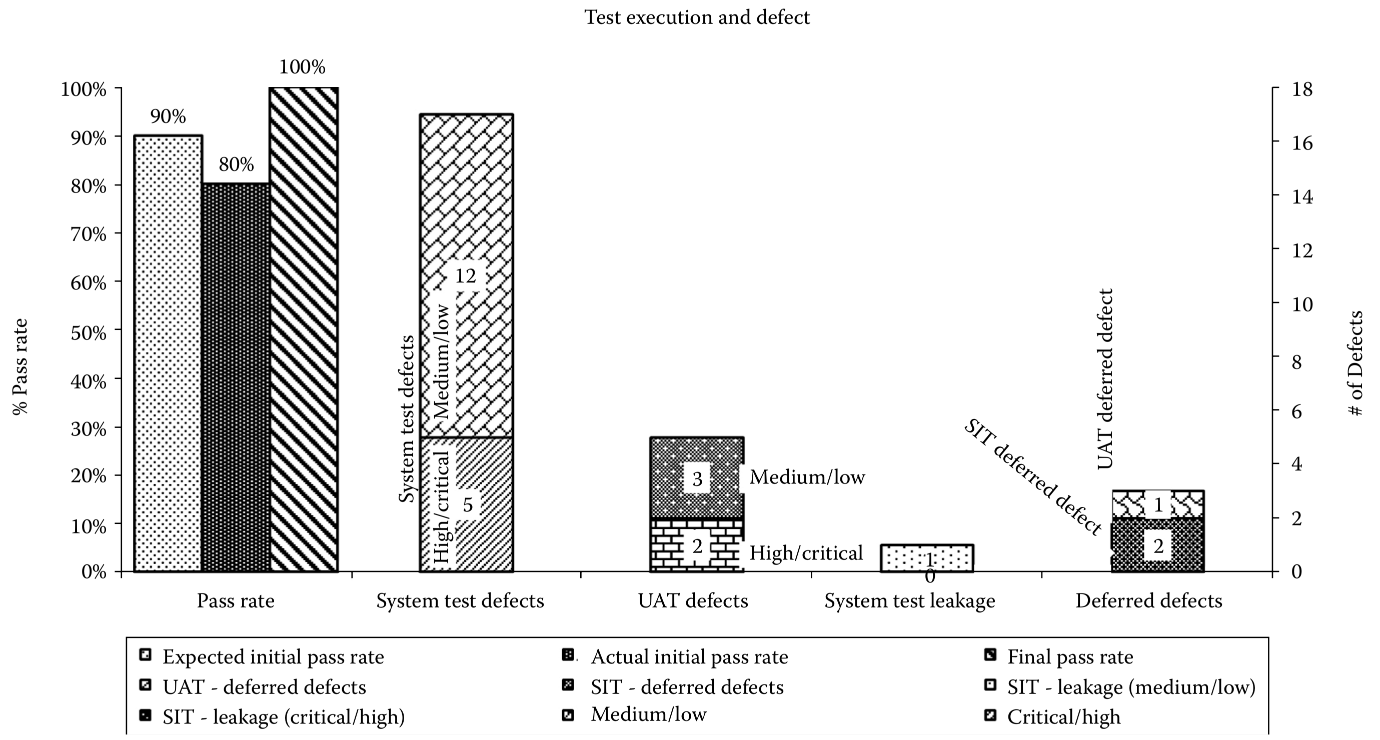

Figure 7.3 A sample report % of pass and defects.

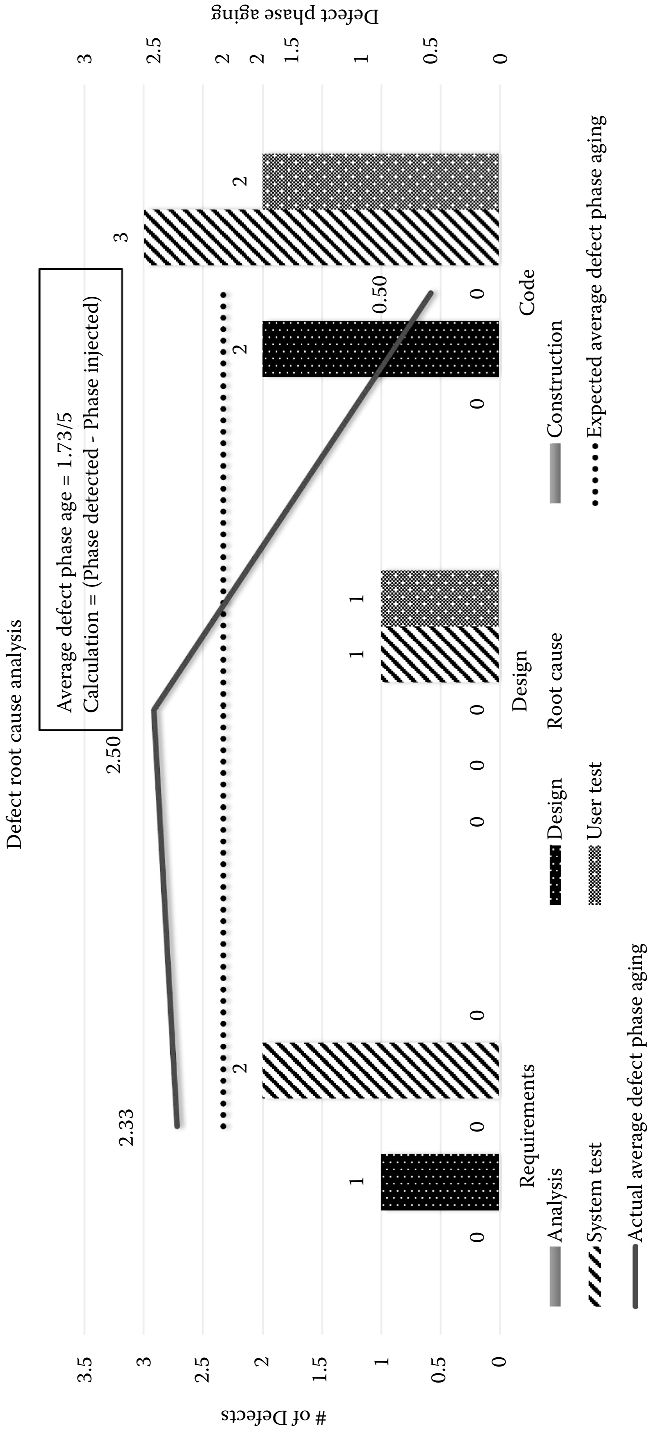

Figure 7.4 Defect root cause analysis, average phase age.

before it reaches its severity level. Validation, verification, inspection, and review helps to prevent defects or severe risk issues.

Therefore, it is imperative to introduce defect prevention at every level of SDLC to prevent defects at the earliest occurrence.

Defect Prediction

Defect prediction is a technique of identifying the quality of the software before deployment. It improves the performance. The main objective is Software Quality Assurance.