Software Testing and Quality Assurance

Theory and Practice

Chapter 13

System Test Execution

Outline of the Chapter

- Modeling Defects
- Preparedness to Start System Testing
- Metrics for Tracking System Test
  - Metrics for Monitoring Test execution
  - Metrics for Monitoring Defect Reports
- Orthogonal Defect Classification
- Defect Causal Analysis
- Beta Testing
- First Customer Shipment
- System Test Report
- Product Sustaining
- Measuring Test Effectiveness
  - Fault Seeding
  - Spoilage Metric

Modeling Defects

- A life-cycle model in the form of a state-transition diagram as shown in Figure 13.1.
- The different states are briefly explained in Table 13.1
- Two key concepts involved in modeling defects are the levels of *priority* and *severity*
- A *priority* level is a measure of how soon the defect needs to be fixed, i.e., urgency.
  - Critical (1), High (2), Medium (3), and Low(4)
- A *severity* level is a measure of the extent of the detrimental effect of the defect on the operation of the product
  - Critical (1), High (2), Medium (3), and Low (4)
- The defect schema is given in Table 13.2

Modeling Defects

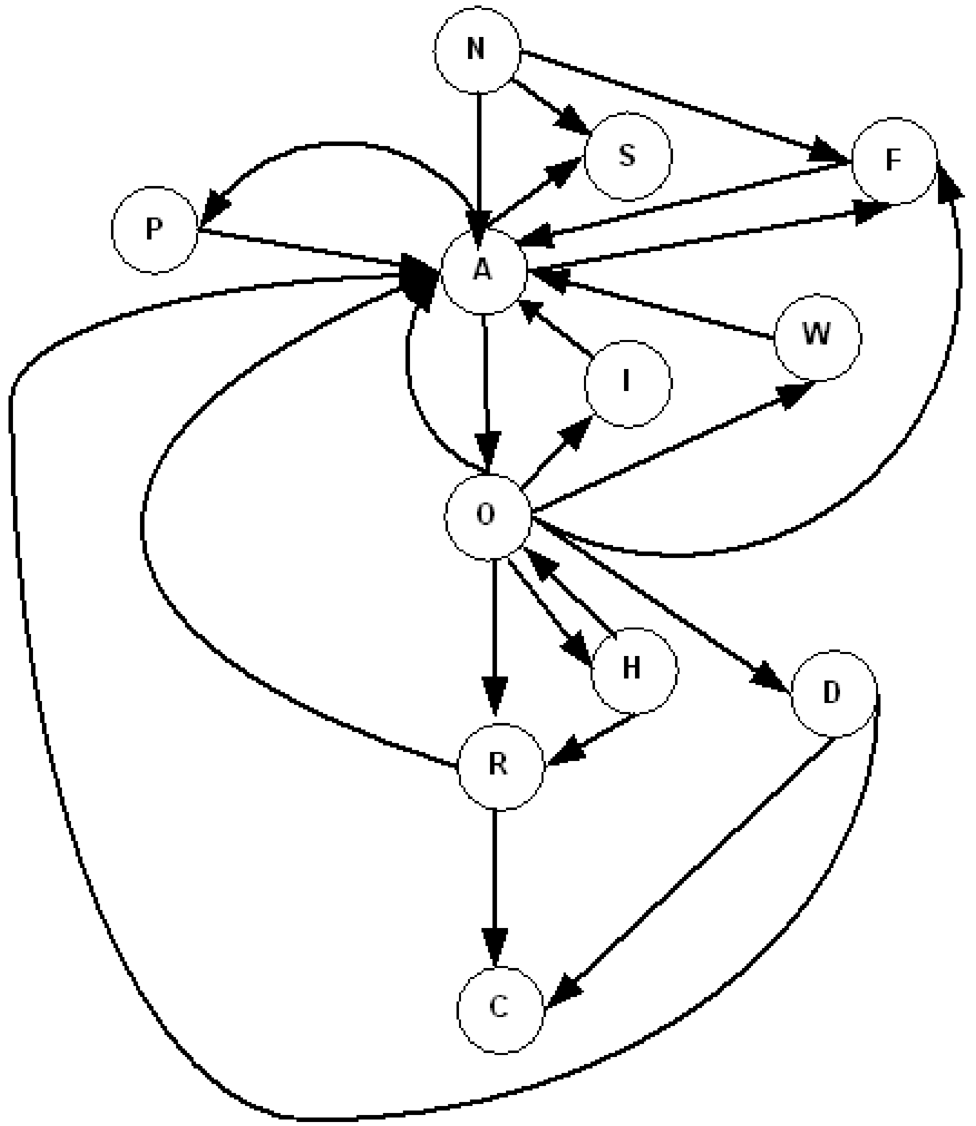
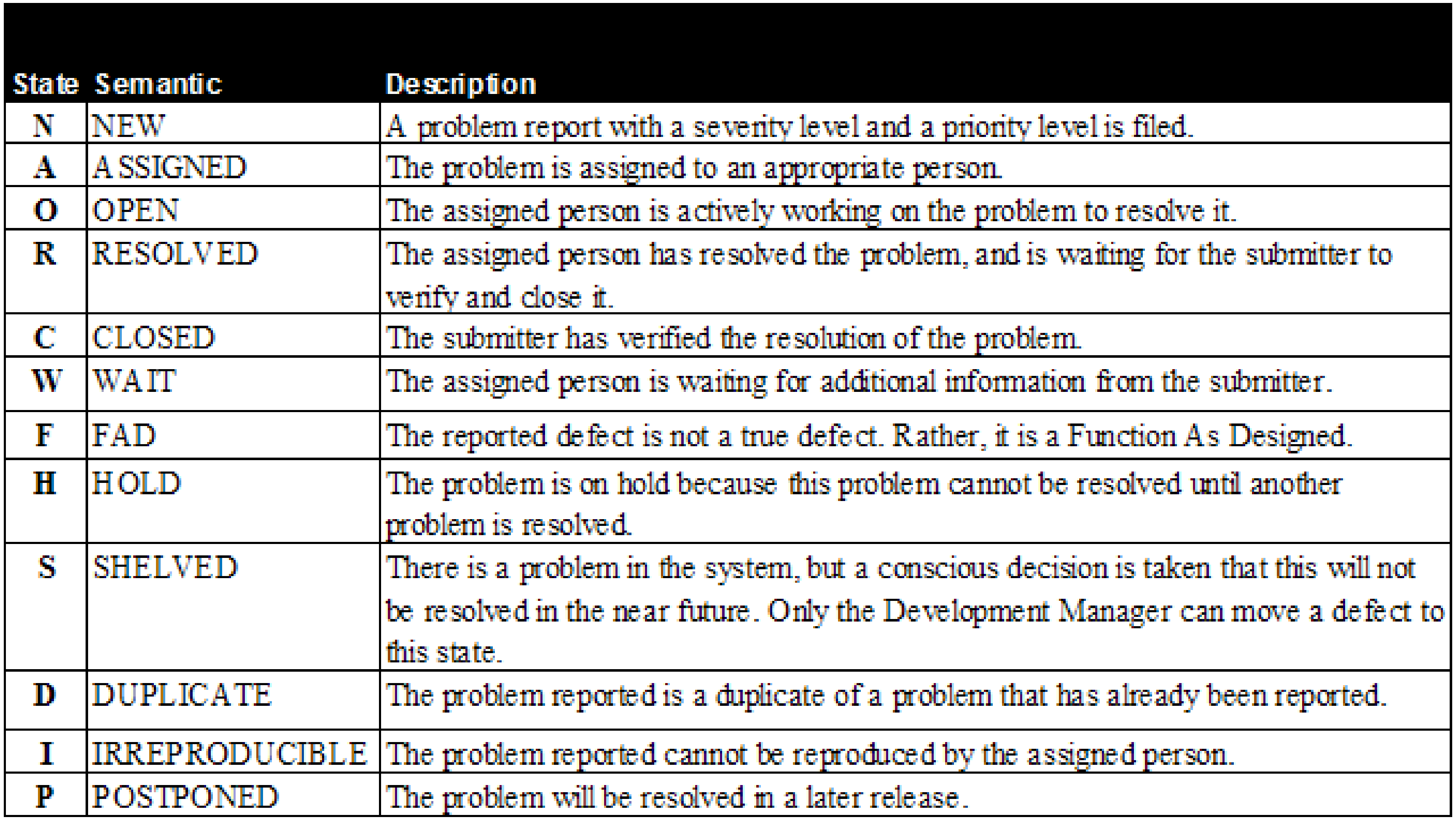


Figure 13.1: State-transition diagram representation of the life-cycle of a defect

Modeling Defects

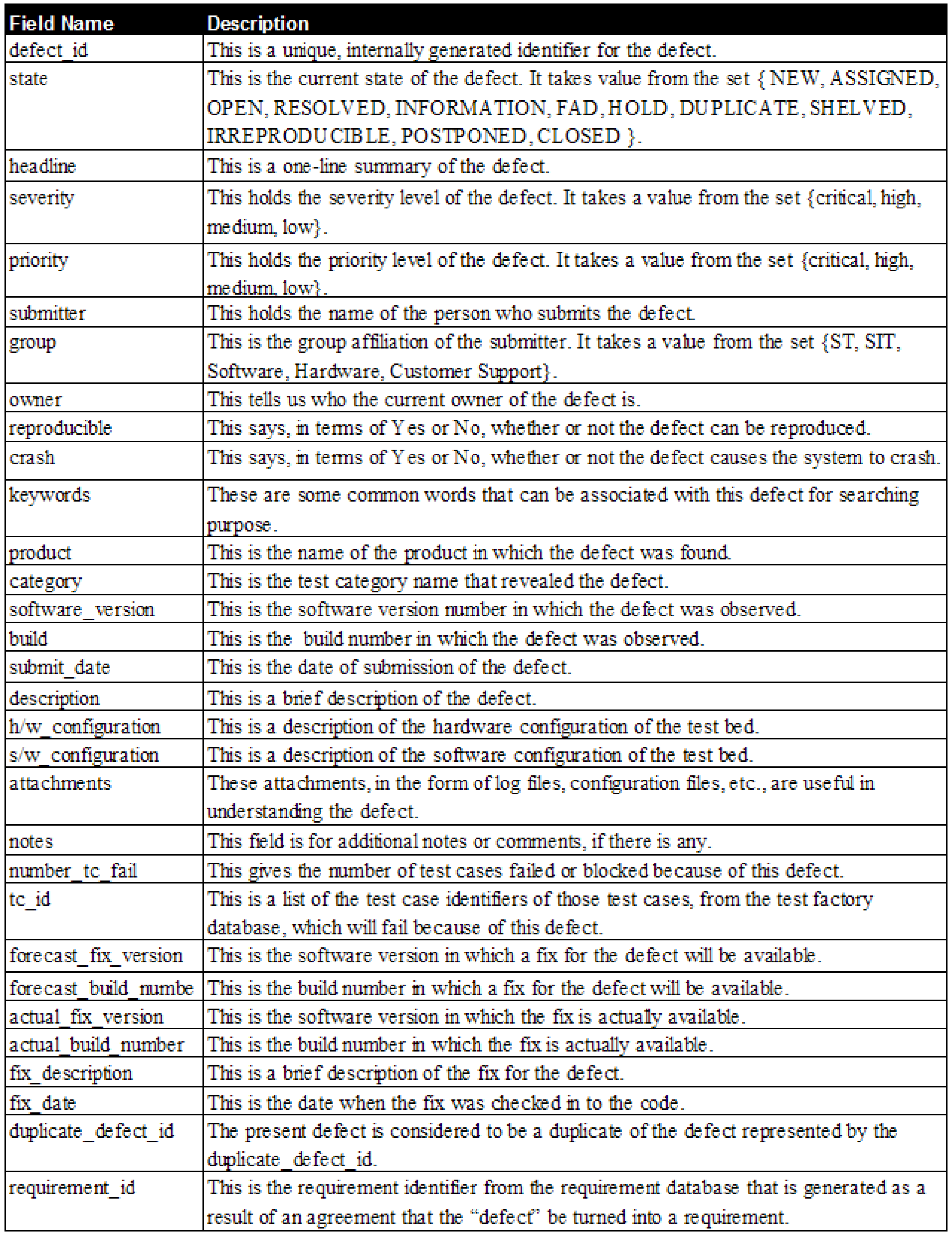


State	Semantic	Description
N	NEW	A problem report with a severity level and a priority level is filed.
A	ASSIGNED	The problem is assigned to an appropriate person.
O	OPEN	The assigned person is actively working on the problem to resolve it.
R	RESOLVED	The assigned person has resolved the problem, and is waiting for the submitter to verify and close it.
C	CLOSED	The submitter has verified the resolution of the problem.
W	WAIT	The assigned person is waiting for additional information from the submitter.
F	FAD	The reported defect is not a true defect. Rather, it is a Function As Designed.
H	HOLD	The problem is on hold because this problem cannot be resolved until another problem is resolved.
S	SHELVED	There is a problem in the system, but a conscious decision is taken that this will not be resolved in the near future. Only the Development Manager can move a defect to this state.
D	DUPLICATE	The problem reported is a duplicate of a problem that has already been reported.
I	IRREPRODUCIBLE	The problem reported cannot be reproduced by the assigned person.
P	POSTPONED	The problem will be resolved in a later release.

Table 13.1: State of a defect modeled in Figure 13.1

Modeling Defects

Table 13.2: Defect schema summary fields



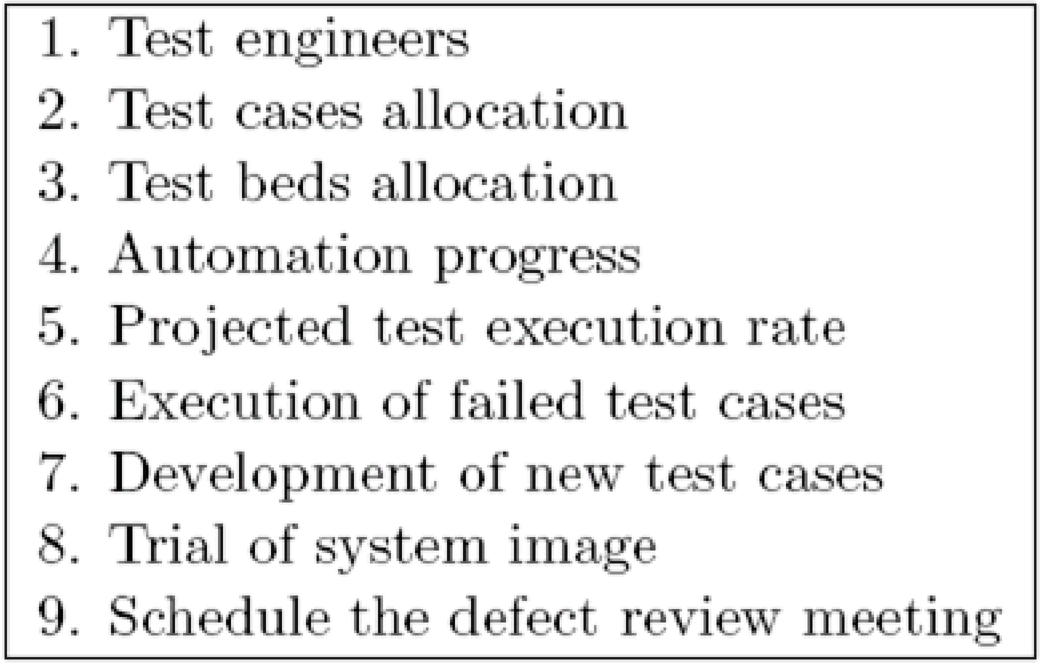
Field Name	Description
defect_id	This is a unique, internally generated identifier for the defect.
state	This is the current state of the defect. It takes value from the set { NEW, ASSIGNED, OPEN, RESOLVED, INFORMATION, FAD, HOLD, DUPLICATE, SHELVED, IRREPRODUCIBLE, POSTPONED, CLOSED }.
headline	This is a one-line summary of the defect.
severity	This holds the severity level of the defect. It takes a value from the set {critical, high, medium, low}.
priority	This holds the priority level of the defect. It takes a value from the set {critical, high, medium, low}.
submitter	This holds the name of the person who submits the defect.
group	This is the group affiliation of the submitter. It takes a value from the set {ST, SIT, Software, Hardware, Customer Support}.
owner	This tells us who the current owner of the defect is.
reproducible	This says, in terms of Yes or No, whether or not the defect can be reproduced.
crash	This says, in terms of Yes or No, whether or not the defect causes the system to crash.
keywords	These are some common words that can be associated with this defect for searching purpose.
product	This is the name of the product in which the defect was found.
category	This is the test category name that revealed the defect.
software_version	This is the software version number in which the defect was observed.
build	This is the build number in which the defect was observed.
submit_date	This is the date of submission of the defect.
description	This is a brief description of the defect.
h/w_configuration	This is a description of the hardware configuration of the test bed.
s/w_configuration	This is a description of the software configuration of the test bed.
attachments	These attachments, in the form of log files, configuration files, etc., are useful in understanding the defect.
notes	This field is for additional notes or comments, if there is any.
number_tc_fail	This gives the number of test cases failed or blocked because of this defect.
tc_id	This is a list of the test case identifiers of those test cases, from the test factory database, which will fail because of this defect.
forecast_fix_version	This is the software version in which a fix for the defect will be available.
forecast_build_numbe	This is the build number in which a fix for the defect will be available.
actual_fix_version	This is the software version in which the fix is actually available.
actual_build_number	This is the build number in which the fix is actually available.
fix_description	This is a brief description of the fix for the defect.
fix_date	This is the date when the fix was checked in to the code.
duplicate_defect_id	The present defect is considered to be a duplicate of the defect represented by the duplicate_defect_id.
requirement_id	This is the requirement identifier from the requirement database that is generated as a result of an agreement that the “defect” be turned into a requirement.

NEC
Empowered by Innovation

Preparedness to Start System Testing

University of Waterloo

- The last item of the entry criteria namely *test execution working document is in place and complete* in Chapter 12, before the start of the system test
- A framework of such document is outlined in Table 13.4



1. Test engineers
2. Test cases allocation
3. Test beds allocation
4. Automation progress
5. Projected test execution rate
6. Execution of failed test cases
7. Development of new test cases
8. Trial of system image
9. Schedule the defect review meeting

Table 13.4: An outline of a test execution working document.

Metrics for Tracking System Test

We categorized execution metrics into two classes:

- Metrics for monitoring test execution
- Metrics for monitoring defects

Metrics for Monitoring Test Execution

- Test case Escapes (TCE)
  - A significant increase in the number of test case escapes implies that deficiencies in the test design
- Planned versus Actual Execution (PAE) Rate
  - Compare the actual number of of test cases executed every week with the planned number of test cases
- Execution Status of Test (EST) Cases
  - Periodically monitor the number of test cases lying in different states
    - *Failed, Passed, Blocked, Invalid* and *Untested*
  - Useful to further subdivide those numbers by test categories

Metrics for Monitoring Defect Reports

- Function as Designed (FAD) Count
- Irreproducible Defects (IRD) Count
- Defects Arrival Rate (DAR) Count
- Defects Rejected Rate (DRR) Count
- Defects Closed Rate (DCR) Count
- Outstanding Defects (OD) Count
- Crash Defects (CD) Count
- Arrival and Resolution of Defects (ARD) Count

Orthogonal Defect Classification

- Orthogonal Defect Classification (ODC) is a methodology for rapid capturing of the semantics of each software defect.
- The classification of defects occurs at two different points in time during the life-cycle of a defect
- In the NEW state, the submitter needs to fill out the following ODC attributes or fields:
  - *Activity:* This is the actual activity that was being performed at the time the defect was discovered
  - *Trigger:* The environment or condition that had to exist for the defect to surface
  - *Impact:* This refers to the effect, the defect would have on the customer, if the defect had escaped to the field.

Orthogonal Defect Classification

- The owner needs to fill out the following ODC attributes or fields when the defect is moved to RESOLVED state:
  - Target: The target represents the high-level identity, such as design, code, or documentation, of the entity that was fixed
  - Defect type: The defect type represents the actual correction that was made
  - Qualifier: The qualifier specifies whether the fix was made due to missing, incorrect, or extraneous code
  - Source: The source indicates whether the defect was found in code developed in house, reused from a library, ported from one platform to another, or provided by a vendor
  - Age: The history of the design or code that had the problem. The age specifies whether the defect was found in new, old (base), rewritten, or refixed code.
    - New: The defect is in a new function which was created by and for the current project.
    - Base: The defect is in a part of the product which has not been modified by the current project. The defect was not injected by the current project
    - Rewritten: The defect was introduced as a direct result of redesigning and/or rewriting of old function in an attempt to improve its design or quality.
    - Refixed: The defect was introduced by the solution provided to fix a previous defect.

Orthogonal Defect Classification

- Data assessment of ODC classified data is based on the relationship of the ODC and non-ODC attributes of a defect
- The assessment must be performed by an analyst who is familiar with the project
- Once the feedback is given to s/w development team, they can take actions to prevent defects from recurring
- The ODC analysis can be combined with Pareto analysis to focus on error-prone parts of the software
- Pareto principle can be stated as *concentrate on the vital few and not the trivial many*
- An alternative expression of the principle is to state that 80% of the problems can be fixed with 20% of the effort
- It is generally called the 80/20 rule

Defect Causal Analysis

- In DCA, defects are analyzed to
  - Determine the cause of an error
  - Take actions to prevent similar errors from occurring in the future
  - Means to remove similar defects that may exist in the system or to detect them at the earliest possible point in the software development process
- It is sometimes referred to as defect prevention or root-cause defect analysis (RCA)
- Three key principles to drive defect causal analysis:
  - Reduce the number of defects to improve quality
  - Apply local expertise where defects occur
  - Focus on systematic errors
- DCA focuses on understanding of cause-effect relationship
  - There must be a correlation between the hypothesized cause and the effect
  - The hypothesized cause must precede the effect in time
  - The mechanism linking the cause to the effect must be identified

Defect Causal Analysis

Linking the causes to the effect consists of five steps:

- When was the failure detected and the corresponding fault injected?
- What scheme is used to classify errors?
- What is the common problem (systematic error)?
- What is the principal cause of the errors?
  - Cause-effect diagram may be used to identify the cause
- How could the error be prevented in the future?
  - Preventive: A preventive action reduces the chances of similar problems occurring in the future
  - Corrective: A corrective action means fixing the problems. Attacking the cause itself may not be cost effective in all situations
  - Mitigating: A mitigating action tries to counter the adverse consequences of problems
- Some Preventive and Corrective actions:
  - Training, Improvement in Communication, Using tools, Process Improvement

Defect Causal Analysis

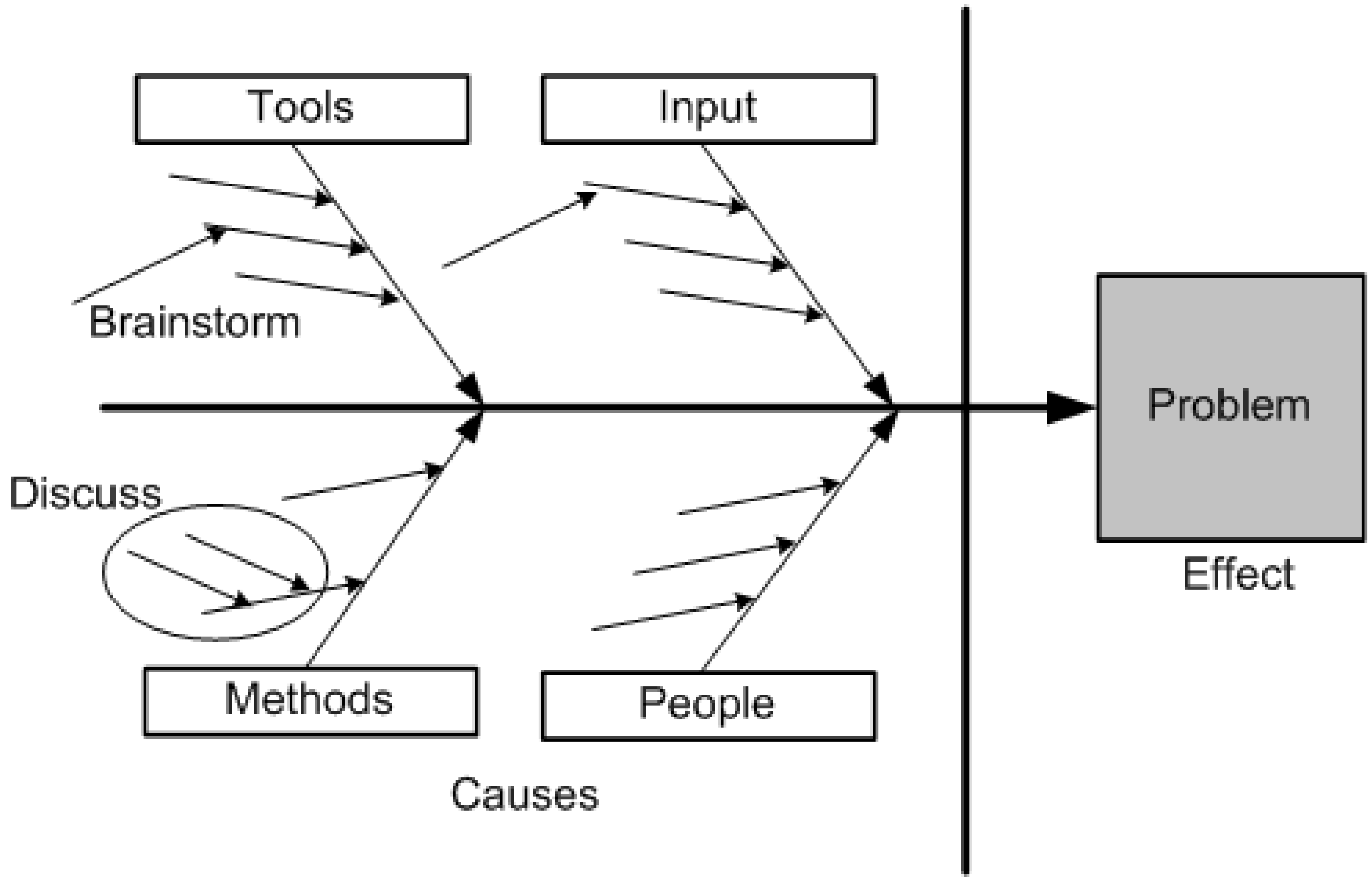
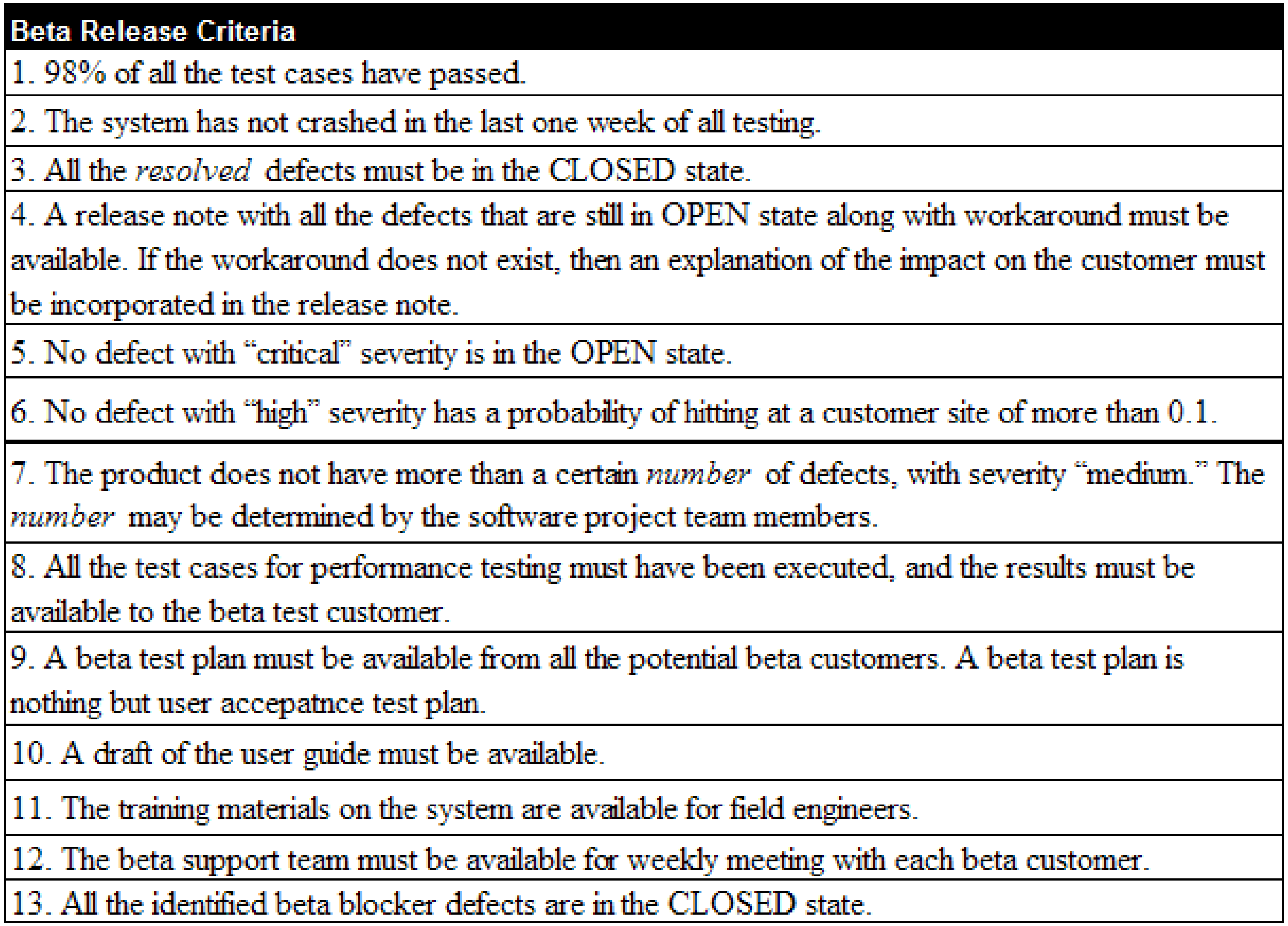


Figure 13.5: Cause-effect diagram for defect causal analysis (DCA).

Beta Testing

- Beta testing is conducted by the potential buyers prior to the official release of the product
- The purpose of beta testing is not to find defects, but to obtain feedback from the field about the usability of the product
- Three kinds of beta:
  - Marketing beta
  - Technical beta
  - Acceptance beta
- A decision about when to release the system to the beta customers is made by the software project team members
- The beta release criteria are established by the project team
- A framework for writing beta release criteria is given in Table 13.13
- The system test cycle continues concurrently with beta testing
- Weekly meetings are conducted with the beta customers by the beta support team to resolve issues.

Beta Testing



Beta Release Criteria
1. 98% of all the test cases have passed.
2. The system has not crashed in the last one week of all testing.
3. All the *resolved* defects must be in the CLOSED state.
4. A release note with all the defects that are still in OPEN state along with workaround must be available. If the workaround does not exist, then an explanation of the impact on the customer must be incorporated in the release note.
5. No defect with "critical" severity is in the OPEN state.
6. No defect with "high" severity has a probability of hitting at a customer site of more than 0.1.
7. The product does not have more than a certain *number* of defects, with severity "medium." The *number* may be determined by the software project team members.
8. All the test cases for performance testing must have been executed, and the results must be available to the beta test customer.
9. A beta test plan must be available from all the potential beta customers. A beta test plan is nothing but user accepatnce test plan.
10. A draft of the user guide must be available.
11. The training materials on the system are available for field engineers.
12. The beta support team must be available for weekly meeting with each beta customer.
13. All the identified beta blocker defects are in the CLOSED state.

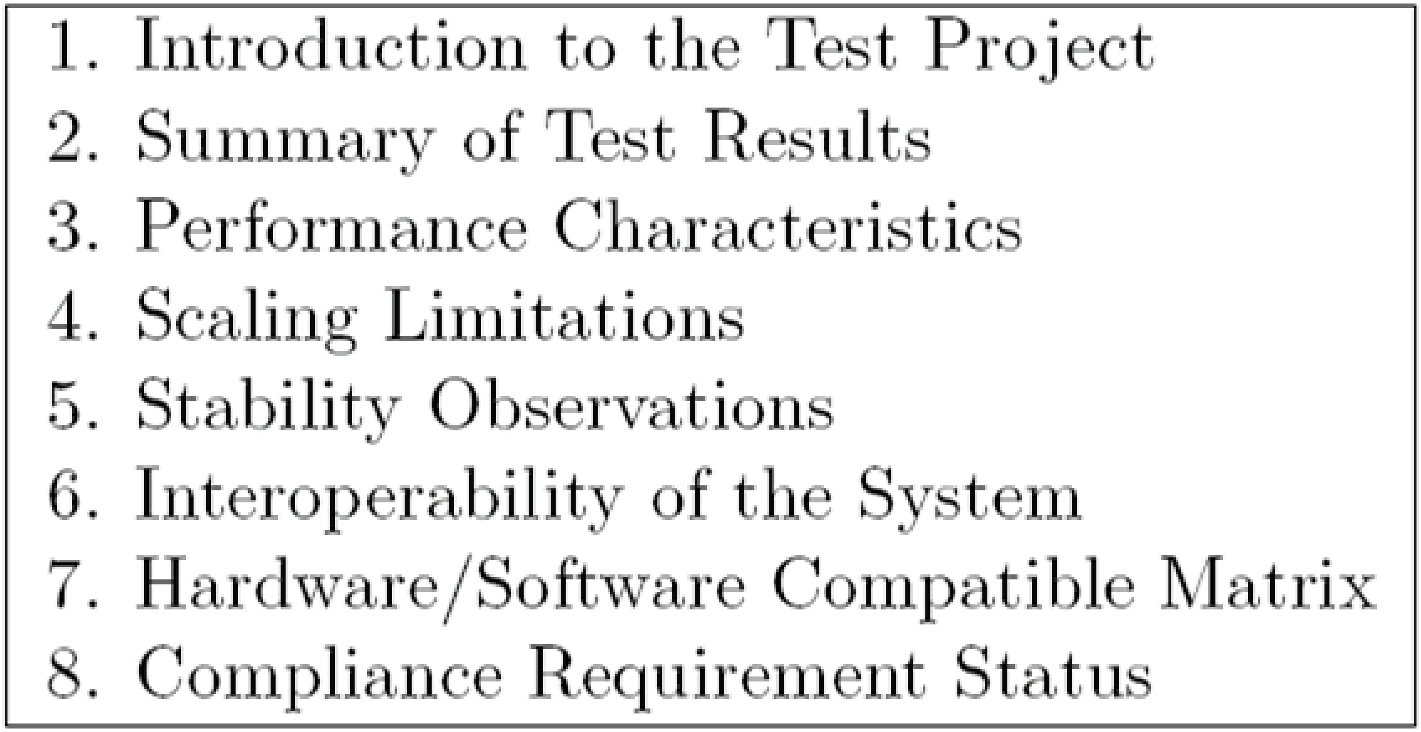
Table 13.13: A framework for beta release criteria

First Customer Shipment

A set of generic FCS readiness criteria is as follows:

- All the test cases from the test suite should have been executed
- Test case results are updated with Passed, Failed, Blocked, or Invalid status
- The requirements are updated by moving each requirement from the Verification state to either the Closed or the Decline state, as discussed in Chapter 11
- The pass rate of test cases is very high, say, 98%
- No crash in the past two weeks of testing has been observed
- No known defect with critical or high severity exists in the product
- Not more than a certain number of known defects with medium and low levels of severity exist in the product
- All the resolved defects must be in the CLOSED state
- The user guides are in place
- Trouble shooting guide is available
- The test report is completed and approved

System Test Report



1. Introduction to the Test Project
2. Summary of Test Results
3. Performance Characteristics
4. Scaling Limitations
5. Stability Observations
6. Interoperability of the System
7. Hardware/Software Compatible Matrix
8. Compliance Requirement Status

Table 13.14: Structure of the final system test report.

Product Sustaining

- Once the product is shipped to one of the paying customer the software project is moved to sustaining phase
- The goal of this phase is to maintain the software quality throughout the product's market life
- Software maintenance activities occur because software testing cannot uncover all the defects in a large software system
- The following three software maintenance activities were coined by Swanson:
  - *Corrective:* The process that includes isolation and correction of one or more defects in the software
  - *Adaptive:* The process that modifies the software to properly interface with a changing environment such as new version of hardware or third party software
  - *Perfective:* The process that improves the software by the addition of new functionalities, enhancements, and/or modifications to the existing functions

Product Sustaining

- The first major task is to determine the type of maintenance task to be conducted when a defect report comes in from a customer
- The **sustaining** team, which include developers and testers are assigned immediately to work on the defect
- If the defect reported by the customer is considered as *corrective* in nature
  - The status of the progress is updated to the customer within 24 hours
  - The group continues to work until a patch with the fix is released to the customer
- If the defect reported is considered as either *adaptive* or *perfective* in nature
  - it is entered in the requirement database, and it goes though the usual software development phases

Product Sustaining

The major tasks of sustaining test engineers are as follows:

- Interact with customer to understand their real environment
- Reproduce the issues observed by the customer in the laboratory environment
- Once the root cause of the problem is known, develop new test cases to verify it
- Develop upgrade/downgrade test cases from the old image to the new patch image
- Participate in the code review process
- Select a subset of regression tests from the existing pool of test cases to ensure that there is no side effect because of the fix
- Execute the selected test cases
- Review the release notes for correctness
- Conduct experiments to evaluate test effectiveness

Measuring Test Effectiveness

- Defect Removal Efficiency (DRE) metric defined as follows:

$$\text{DRE} = \frac{\text{Number of Defects Found in Testing}}{\text{Number of Defects Found in Testing} + \text{Number of Detects Not Found}}$$

Fault Seeding

- Fault seeding approach is used to estimate the # of escaped defects
- Inject a small number of defects into system
- Measure the percentage of defects that are uncovered by the test engineers
- Estimate the number of actual defects using an extrapolation technique

Suppose that the product contains N defects, and K defects are seeded

At the end of the test experiments the team has found n unseeded and

k seeded defects.

The fault seeding theory asserts the following

$$\frac{k}{K} = \frac{n}{N} \Rightarrow N = n\left(\frac{K}{k}\right)$$

Spoilage Metric

- Defects are injected and removed at different phases of a software development cycle
- The cost of each defect injected in phase X and removed in phase Y increases with the increase in the distance between X and Y
- An effective testing method would find defects earlier than a less effective testing method would
- An useful measure of test effectiveness is defect age, called PhAge

Phase Injected	Phase Discovered							
	Requirements	High-Level Design	Detailed Design	Coding	Unit Testing	Integration Testing	System Testing	Acceptance Testing
Requirements	0	1	2	3	4	5	6	7
High-Level Design		0	1	2	3	4	5	6
Detailed Design			0	1	2	3	4	5
Coding				0	1	2	3	4

Table 13.15: Scale for defect age

Spoilage Metric

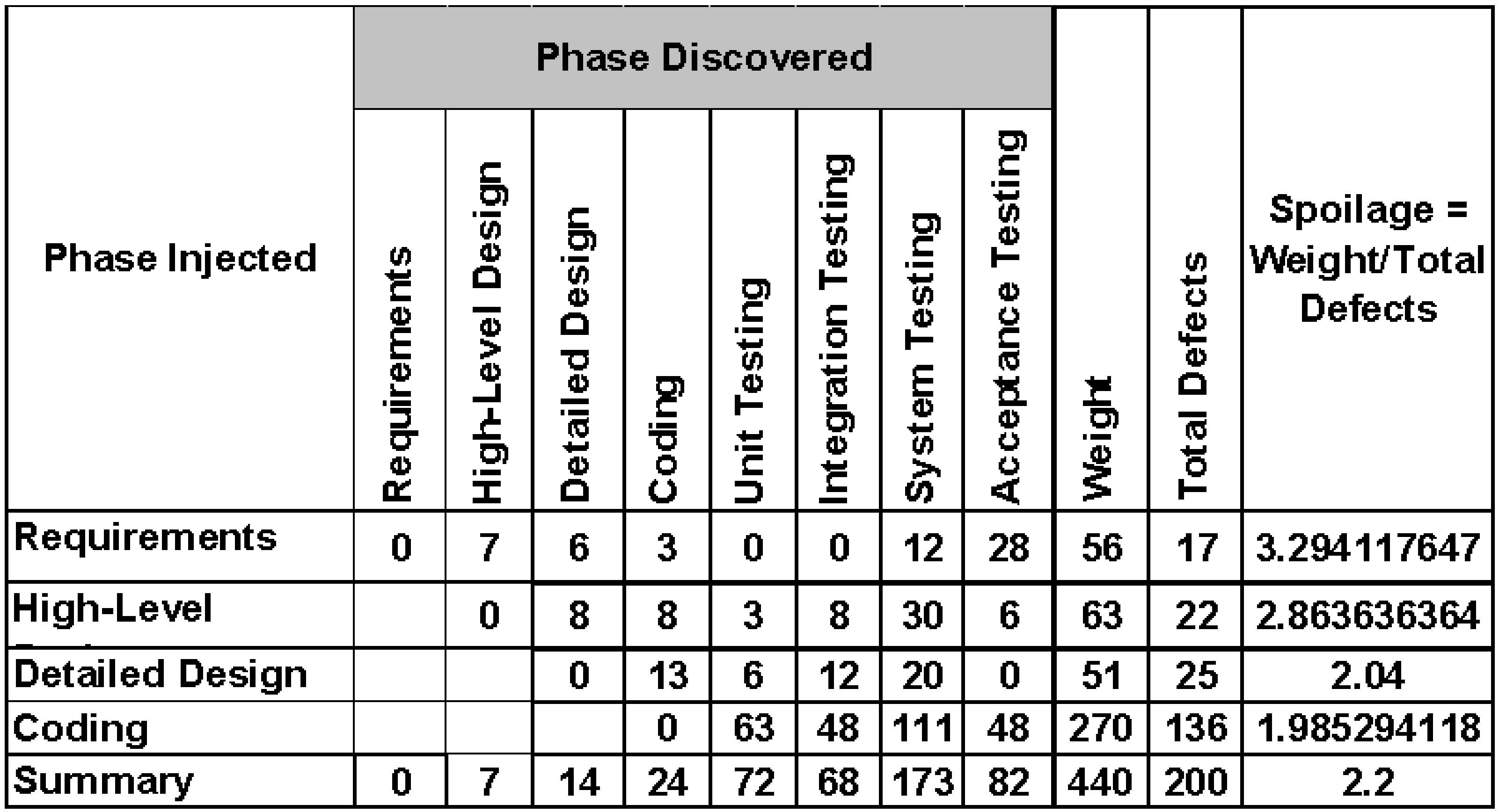
Phase Injected	Phase Discovered								Total Defects
	Requirements	High-Level Design	Detailed Design	Coding	Unit Testing	Integration Testing	System Testing	Acceptance Testing	
Requirements	0	7	3	1	0	0	2	4	17
High-Level Design		0	8	4	1	2	6	1	22
Detailed Design			0	13	3	4	5	0	25
Coding				0	63	24	37	12	136
Summary	0	7	11	18	67	30	50	17	200

Table 13.16: Defect injection and versus discovery on project *Boomerang*

A new metric called spoilage is defined as

$$\text{Spoliage} = \frac{\sum(\text{Number of Defects} \times \text{Discovered Phage})}{\text{Total Number of Defects}}$$

Spoilage Metric



Phase Injected	Phase Discovered								Weight	Total Defects	Spoilage = Weight/Total Defects
	Requirements	High-Level Design	Detailed Design	Coding	Unit Testing	Integration Testing	System Testing	Acceptance Testing			
Requirements	0	7	6	3	0	0	12	28	56	17	3.294117647
High-Level		0	8	8	3	8	30	6	63	22	2.863636364
Detailed Design			0	13	6	12	20	0	51	25	2.04
Coding				0	63	48	111	48	270	136	1.985294118
Summary	0	7	14	24	72	68	173	82	440	200	2.2

Table 13.17: Number of defects weighted by defect age on project *Boomerang*

Spoilage Metric

- The spoilage value for the Boomerang test project is 2.2
- A spoilage value close to 1 is an indication of a more effective defect discovery process
- As an absolute value, the spoilage metric has little meaning
- This metric is useful in measuring the long-term trend of test effectiveness in an organization