Chapter 8

SOFTWARE-TESTING STRATEGIES

Achmad Benny Mutiara
amutiara@staff.gunadarma.ac.id

8.1 STATIC-TESTING STRATEGIES

- Static testing is the systematic examination of a program structure for the purpose of showing that certain properties are true regardless of the execution path the program may take.
  - Consequently, some static analyses can be used to demonstrate the absence of some faults from a program.
- Static testing represents actual behavior with a model based upon the program's semantic features and structure.
  - Human comparison often consists of people exercising little discipline in comparing their code against notions of intent that are only loosely and imprecisely defined. But human comparisons may also be quite structured, rigorous, and effective as is the case of inspections and walkthroughs, which are carefully defined and administered processes orchestrating groups of people to compare code and designs to careful specifications of intent.

- Static testing strategies include:
  - Formal technical reviews
  - Walkthroughs
  - Code inspections
  - Compliance with design and coding standards

8.1.1 Formal Technical Reviews

- A review can be defined as:
  - *A meeting at which the software element is presented to project personnel, managers, users, customers, or other interested parties for comment or approval.*
- What is a software review? A software review can be defined as a filter for the software-engineering process.
- The purpose of any review is
  - to discover errors in the analysis, design, and coding, testing and implementation phases of the software development cycle.
  - to see whether procedures are applied uniformly and in a manageable manner.

Objectives for Reviews

- Review objectives are used:
  - To ensure that the software elements conform to their specifications.
  - To ensure that the development of the software element is being done as per
  - plans, standards, and guidelines applicable for the project.
  - To ensure that the changes to the software elements are properly implemented
  - and affect only those system areas identified by the change specification.

Types of Reviews

- Reviews are one of two types: informal technical reviews and formal technical reviews.
  - **Informal Technical Review: An informal meeting and informal desk** checking.
  - **Formal Technical Review (FTR): A formal software quality assurance activity** through various approaches, such as structured walkthroughs, inspections, etc.

What is a Formal Technical Review (FTR)?

- A formal technical review (FTR) is a software quality assurance activity performed by software-engineering practitioners to improve software product quality.
  - The product is scrutinized for completeness, correctness, consistency, technical feasibility, efficiency, and adherence to established standards and guidelines by the client organization.
- The FTR serves as a training ground, enabling junior engineers to observe different approaches to software analysis, design, and implementation. Each FTR is conducted as a meeting and will be successful only if it is properly planned, controlled, and attended.

Objectives of a Formal Technical Review

- The various objectives of a formal technical review are as follows:
  - To uncover errors in logic or implementation.
  - To ensure that the software has been represented according to predefined standards.
  - To ensure that the software under review meets the requirements.
  - To make the project more manageable.
- For the success of a formal technical review, the following are expected:
  - The schedule of the meeting and its agenda reach the members well in advance.
  - Members review the material and its distribution.
  - The reviewer must review the material in advance.

The Review Meeting

- The meeting should consist of two to five people and should be restricted to not more than two hours (preferably).
- The aim of the review is to review the product/work and the performance of people.
  - When the product is ready, the producer (developer) informs the project leader about the completion of the product and requests for review. The project leader contacts the review leader for the review.
  - The review leader asks the reviewer to perform an independent review of the product/work before the scheduled FTR.

Results of FTR

- Meeting decision
  1. Whether to accept the product/work without any modifications.
  2. Accept the product/work with certain changes.
  3. Reject the product/work due to error.
- Review summary report
  1. What was reviewed?
  2. Who reviewed it?
  3. Findings of the review.
  4. Conclusion.

8.1.2 Code Walk-throughs

- A code walk-through is an informal analysis of code as a cooperative, organized activity by several participants.
- The analysis is based mainly on the game of “playing the computer.”
  - That is, participants select some test cases (the selection could have been done previously by a single participant) and simulate execution of the code by hand.
  - This is the reason for the name walk-through: participants “walk through the code” or through any design notation.

In general, the following prescriptions are recommended:

- Everyone's work should be reviewed on a scheduled basis.
- The number of people involved in the review should be small (three to five).
- The participants should receive written documentation from the designer a few days before the meeting.
- The meeting should last a predefined amount of time (a few hours).
- Discussion should be focused on the discovery of errors, not on fixing them, nor on proposing alternative design decisions.
- Key people in the meeting should be the designer, who presents and explains the rationale of the work, a moderator for the discussion, and a secretary, who is responsible for writing a report to be given to the designer at the end of the meeting.
- In order to foster cooperation and avoid the feeling that the designers are being evaluated, managers should not participate in the meeting.

8.1.3 Code Inspections

- A code inspection, originally introduced by Fagan (1976) at IBM, is similar to a walk-through but is more formal. In Fagan's experiment, three separate inspections were performed: one following design, but prior to implementation; one following implementation, but prior to unit testing; and one following unit testing. The inspection following unit testing was not considered to be cost effective in discovering errors; therefore, it is not recommended.
- The organization aspects of code inspection are similar to those of code walkthrough (i.e., the number of participants, duration of the meeting, psychological attitudes of the participants, etc., should be about the same), but there is a difference in goals.
- In code inspection, the analysis is aimed explicitly at the discovery of commonly made errors. In such a case, it is useful to state beforehand the type of errors for which we are searching. For instance, consider the classical error of writing a procedure that modifies a formal parameter and calling the procedure with a constant value as the actual parameter.

The following is a list of some classical programming errors, which can be checked for during code inspection:

- Use of uninitialized variables
- Jumps into loops
- Non-terminating loops
- Incompatible assignments
- Array indices out of bounds
- Improper storage allocation and deallocation
- Mismatches between actual and formal parameters in procedure calls
- Use of incorrect logical operators or incorrect precedence among operators
- Improper modification of loop variables
- Comparison of equality of floating-point values, etc.

Checklist for Code Inspections

- Inspections or reviews are more formal and conducted with the help of some kind of checklist. The steps in the inspections or reviews are:
  - Is the number of actual parameters and formal parameters in agreement?
  - Do the type attributes of actual and formal parameters match?
  - Do the dimensional units of actual and formal parameters match?
  - Are the number of attributes and ordering of arguments to built-in functions correct?
  - Are constants passed as modifiable arguments?
  - Are global variable definitions and usage consistent among modules?
  - Application of a checklist specially prepared for the development plan, SRS, design and architecture
  - Nothing observation: ok, not ok, with comments on mistake or inadequacy Repair-rework
  - Checklists prepared to countercheck whether the subject entity is correct, consistent, and complete in meeting the objectives

8.1.4 Differences Between Walk-throughs and Inspections/ Reviews

- The basic difference between the two is that a walk-through is less formal and has only a few steps, whereas inspections and reviews are more formal and logically sequential with many steps.
- Both processes are undertaken before actual development, and hence they are conducted on documents, such as a development plan, SOW, RDD and SRS, design document, and broad WBS to examine their authenticity, completeness, correctness, and accuracy.
- Both are costly but the cost incurred is comparatively much lower than the cost of repair at a much later stage in the development cycle.
- Another difference between a walk-through and an inspection is that the former is less formal and quick; whereas inspection is more formal, takes more time, and is far more systematic.

8.2 DEBUGGING

8.2.1 Introduction/Definition

- Debugging means identifying, locating, and correcting the bugs usually by running the program. It is an extensively used term in programming. These bugs are usually logical errors.
  - During the compilation phase the source files are accessed and if errors are found, then that file is edited and the corrections are posted in the file. After the errors have been detected and the corrections have been included in the source file, the file is recompiled.
  - This detection of errors and removal of those errors is called debugging.
  - The file is compiled again, so changes done last time get included in the object file also by itself.
  - This process of compilation, debugging, and correction posting in the source file continues until all syntactical errors are removed completely.
  - If a program is very large and complex, the more the program has to be corrected and compiled.
- Successful compilation of the program means that now the program is following all the rules of the language and is ready to execute. All of the syntax errors of the program are indicated by the complier at this stage.

8.2.2 Debugging Tactics/Categories

- The various categories for debugging are:
  - Brute-force debugging
  - Backtracking
  - Cause elimination
  - Program slicing
  - Fault-tree analysis

The various categories for debugging mentioned above are discussed as follows:

- ***Brute-force Debugging. The programmer appends the print or write statement*** which, when executed, displays the value of a variable. The programmer may trace the value printed and locate the statement containing the error. Earlier when the time for execution was quite high, programmers had to use the core dumps. The core dumps are referred to as the static image of the memory and this may be scanned to identify the bug.

- ***Backtracking. In this technique, the programmer backtracks from the place or*** statement which gives the error symptoms for the first time. From this place, all the statements are checked for possible cause of errors. Unfortunately, as the number of source lines increases, the number of potential backward paths may become unmanageably large.

- ***Cause Elimination. Cause elimination is manifested by induction or deduction*** and introduces the concept of binary partitioning. Data related to the error occurrence are organized to isolate potential causes. A list of all possible causes is developed and tests are conducted to eliminate each. If initial tests indicate that a particular cause hypothesis shows promise, the data are refined in an attempt to isolate the bug.

- ***Program Slicing. This technique is similar to backtracking. However, the*** search space is reduced by defining slices. A slice of a program for a particular variable at a particular statement is the set of source lines preceding this statement that can influence the value of that variable.

- ***Fault-tree Analysis. Fault-tree analysis, a method originally developed for the*** U.S. Minuteman missile program, helps us to decompose the design and look for situations that might lead to failure. In this sense, the name is misleading; we are really analyzing failures, not faults, and looking for potential causes of those failures. We build fault trees that display the logical path from effect to cause. These trees are then used to support fault correction or tolerance, depending on the design strategy we have chosen.

8.2.3 Debugging Process

- Debugging is not testing but always occurs as a consequence of testing.
- Referring to Figure 8.1, the debugging process begins with the execution of a test case. Results are assessed and a lack of correspondence between expected and actual performance is encountered.
  - In many cases, the lack of corresponding data is a symptom of an underlying cause as still hidden.
  - Debugging attempts to match symptom with cause, thereby leading to error correction.
- Debugging will always have one of two outcomes:
  - The cause will be found and corrected and removed or
  - The cause will not be found.

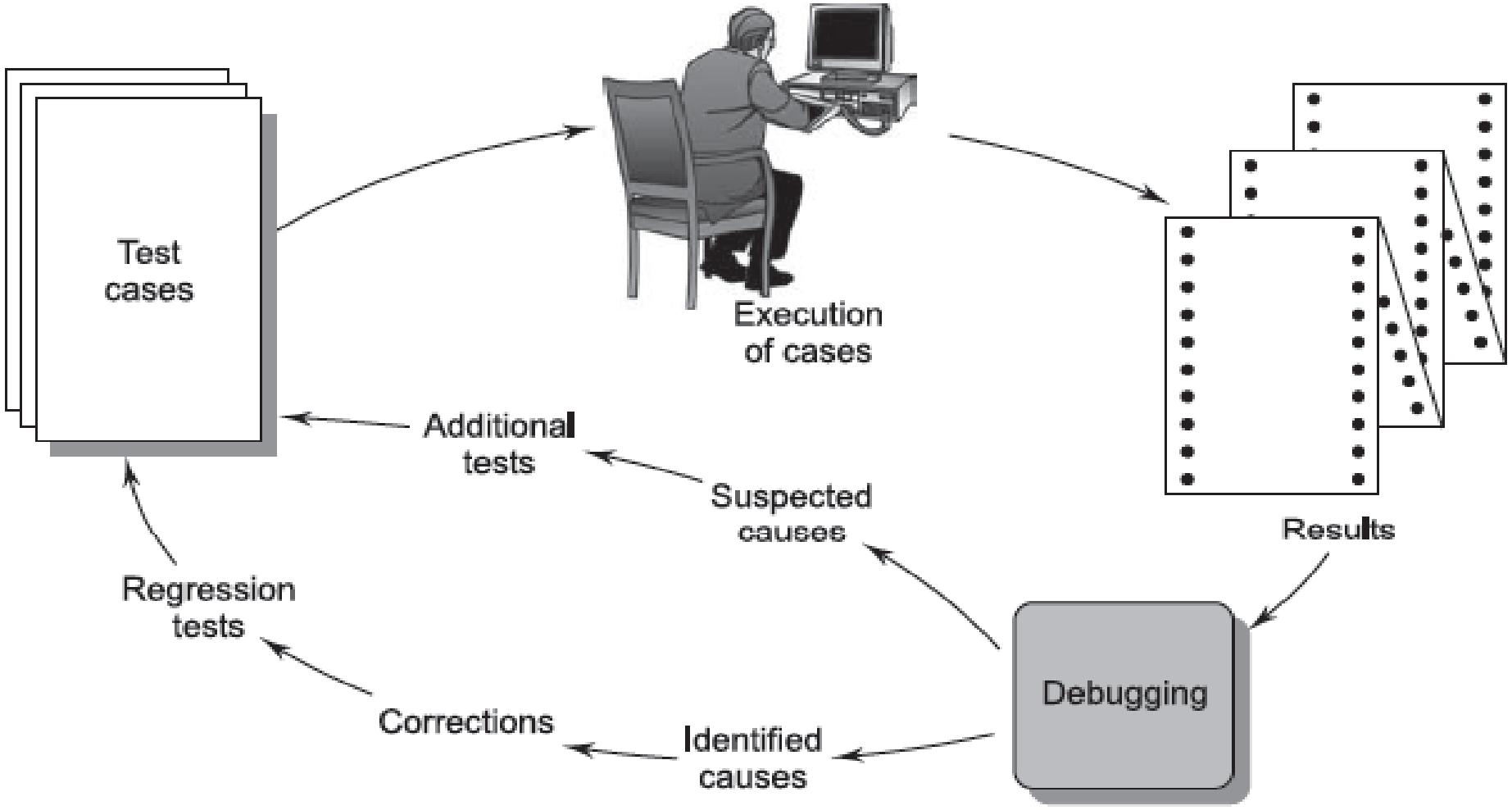

FIGURE 8.1 The Debugging Process

8.2.4 Program Debugging

- **People think that program testing and debugging are the same thing**. Though closely related, they are **two distinct processes**.
  - **Testing** establishes the presence of errors in the program.
  - **Debugging** is the locating of those errors and correcting them. Debugging depends on the output of testing which tells the programmer about the presence or absence of errors.

- There are various debugging stages, as shown in Figure 8.2. The incorrect parts of the code are located and the program is modified to meet its requirements. After repairing, the program is tested again to ensure that the errors have been corrected. Debugging can be viewed as a problem-solving process.

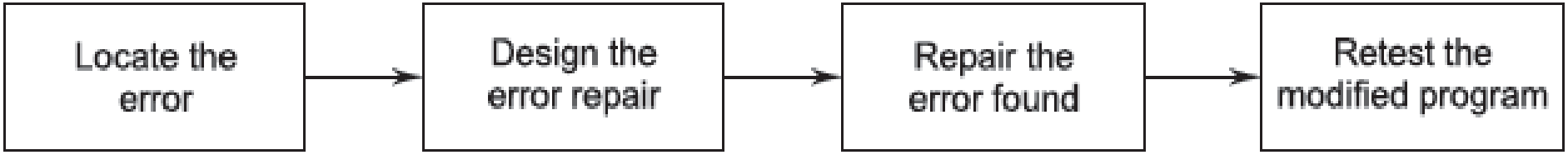

FIGURE 8.2 Debugging Stages

- There is no standard method to teach how to debug a program.
  - The debugger must be a skilled person who can easily understand the errors by viewing the output.
  - The debugger must have knowledge of common errors, which occur very often in a program.
- After errors have been discovered, then correct the error. If the error is a coding error, then that error can be corrected easily. But, if the error is some design mistake, then it may require effort and time. Program listings and the hard copy of the output can be an aid in debugging.

8.2.5 Debugging Guidelines

Some general guidelines for effective debugging include:

- Many a times, debugging requires a thorough understanding of the program design.
- Debugging may sometimes even require a full redesign of the system.
- One must be aware of the possibility that any error correction may introduce new errors. Therefore, after every round of error-fixing, regression testing must be carried out.

8.2.6 Characteristics of Bugs

Some characteristics of bugs are as follows:

- The symptom and the cause may be geographically remote.
- The symptom may disappear when another error is corrected.
- The symptom may actually be caused by non-errors.
- The symptom may be caused by a human error.
- The symptom may be a result of timing problems.
- It may be difficult to accurately reproduce input conditions.
- The symptom may be intermittent.
- The symptom may be due to causes that are distributed across a number of tasks running on different processors.

8.3 ERROR, FAULT, AND FAILURE

8.3.1 Errors

- An error is a discrepancy between the actual value of the output given by the software and the specified correct value of the output for that given input.
  - That is, error refers to the difference between the actual output of the software and the correct output.
- An error is also used to refer to the wrong decision in a given case as compared to what is expected to be the right one.
- Error also refers to human actions that result in software containing a defect or fault.

Types of Errors

Errors can be classified into two categories:

- **Syntax Error. A syntax error is a program statement that violates one or more** rules of the language in which it is written.
- **Logic Error. A logic error deals with incorrect data fields, out-of-range terms,** and invalid combinations.

8.3.2 Faults

- ***A fault is a condition that causes a system to fail in performing its required function.***
- A fault is the basic reason for software malfunction. It is also commonly called a bug. Even though correct input is given to the system, when it fails then we say the system has a fault or a bug, and needs repair.
- The number of faults in software is the difference between the number introduced and the number removed.
- Faults are introduced when the code is being developed by pprogrammers. They may introduce the faults during original design or when they are adding new features, making design changes, or repairing faults that have been identified.

- Faults removal obviously can't occur unless you have some means of detecting the fault in the first place.
  - Thus, fault removal resulting from execution depends on the occurrence of the associated failure. Occurrence depends both on the length of time for which the software has been executing and on the execution environment or operational profile.
- When different functions are executed, different faults are encountered and the failures that are exhibited tend to be different;
  - thus, are environmental influence.
- We can often find faults without execution. They may be found through inspection, compiler diagnostics, design or code reviews, or code reading.

8.3.3 Failure

- ***Failure is the inability of the software to perform a required function to its specification.***

- In other words, when software goes ahead in processing without showing error or fault even though certain input and process specification are violated, then it is called a **software failure**.
  - A software failure occurs when the behavior of software is different from the required behavior.
- A failure is produced only when there is a fault in the system. In other words, faults have the potential to cause failures and their presence is a necessary but not a sufficient condition for failure to occur.

EXERCISES

1. Explain, in brief, the various static-testing strategies.
2. Give a comparative study of inspection, reviews, walk-throughs, and checklists.
3. Define various testing strategies in detail.
4. What is a code walk-through? List the important types of errors checked during a code walk-through.
5. How can design attributes facilitate debugging?
6. What are the various debugging approaches? Discuss them with the help of examples.
7. Define the term “debugging.” Explain the various debugging techniques available.
8. Why is it advantageous to detect as many errors as possible during code review than during testing?
9. Define a review. Also, explain the different types of reviews.
10. What is a Formal Technical Review (FTR)? What are the objectives of a FTR?
11. What is the role of a formal technical review as a quality-assurance activity? Discuss the details of the review meeting, reporting, and record keeping.

12. Enumerate the various steps involved in a inspection/review.
13. What is the difference between a code walk-through and a code inspection/review?
14. What are the guidelines used for effective debugging?
15. Describe the debugging process with the help of a suitable diagram.
16. What is program debugging?
17. Enumerate some of the characteristics shown by bugs.
18. What do you understand by the terms error, fault, and failure?
19. What is the difference between a syntax error and logical error?

Tanya Jawab

The End