Software testing

Lecture – 22

Software Testing

A more appropriate definition is:

"Testing is the process of executing a program with the intent of finding errors."

Software Testing

- **Why should We Test ?**

Although software testing is itself an expensive activity, yet launching of software without testing may lead to cost potentially much higher than that of testing, specially in systems where human safety is involved.

In the software life cycle the earlier the errors are discovered and removed, the lower is the cost of their removal.

Software Testing

- **What is Testing?**

Many people understand many definitions of testing :

1. **Testing is the process of demonstrating that errors are not present.**

2. **The purpose of testing is to show that a program performs its intended functions correctly.**
3. **Testing is the process of establishing confidence that a program does what it is supposed to do.**

These definitions are incorrect.

Software Testing

- **Who should Do the Testing ?**

  - **Testing requires the developers to find errors from their software.**
  - **It is difficult for software developer to point out errors from own creations.**
  - **Many organizations have made a distinction between development and testing phase by making different people responsible for each phase.**

Software Testing

- **What should We Test ?**

We should test the program's responses to every possible input. It means, we should test for all valid and invalid inputs. Suppose a program requires two 8 bit integers as inputs. Total possible combinations are $2^8 x 2^8$. If only one second it required to execute one set of inputs, it may take 18 hours to test all combinations. Practically, inputs are more than two and size is also more than 8 bits. We have also not considered invalid inputs where so many combinations are possible. Hence, complete testing is just not possible, although, we may wish to do so.

Some Terminologies

- **Error, Mistake, Bug, Fault and Failure**

People make errors. A good synonym is mistake. This may be a syntax error or misunderstanding of specifications. Sometimes, there are logical errors.

When developers make mistakes while coding, we call these mistakes "bugs".

A fault is the representation of an error, where representation is the mode of expression, such as narrative text, data flow diagrams, ER diagrams, source code etc. Defect is a good synonym for fault.

A failure occurs when a fault executes. A particular fault may cause different failures, depending on how it has been exercised.

➢ Test, Test Case and Test Suite

Test and Test case terms are used interchangeably. In practice, both are same and are treated as synonyms. Test case describes an input description and an expected output description.

Test Case ID	
Section-I (Before Execution)	Section-II (After Execution)
Purpose :	Execution History:
Pre condition: (If any)	Result:
Inputs:	If fails, any possible reason (Optional);
Expected Outputs:	Any other observation:
Post conditions:	Any suggestion:
Written by:	Run by:
Date:	Date:

Fig. 2: Test case template

The set of test cases is called a test suite. Hence any combination of test cases may generate a test suite.

Software Testing

➢ Verification and Validation

Verification is the process of evaluating a system or component to determine whether the products of a given development phase satisfy the conditions imposed at the start of that phase.

Validation is the process of evaluating a system or component during or at the end of development process to determine whether it satisfies the specified requirements .

Testing= Verification+Validation

Software Testing

➢ Alpha, Beta and Acceptance Testing

The term Acceptance Testing is used when the software is developed for a specific customer. A series of tests are conducted to enable the customer to validate all requirements. These tests are conducted by the end user / customer and may range from adhoc tests to well planned systematic series of tests.

The terms alpha and beta testing are used when the software is developed as a product for anonymous customers.

Alpha Tests are conducted at the developer's site by some potential customers. These tests are conducted in a controlled environment. Alpha testing may be started when formal testing process is near completion.

Beta Tests are conducted by the customers / end users at their sites. Unlike alpha testing, developer is not present here. Beta testing is conducted in a real environment that cannot be controlled by the developer.

Structural Testing

A complementary approach to functional testing is called structural / white box testing. It permits us to examine the internal structure of the program.

Path Testing

Path testing is the name given to a group of test techniques based on judiciously selecting a set of test paths through the program. If the set of paths is properly chosen, then it means that we have achieved some measure of test thoroughness.

This type of testing involves:

1. **generating a set of paths that will cover every branch in the program.**
2. **finding a set of test cases that will execute every path in the set of program paths.**

Hence, our objective is to find all du-paths and then identity those du-paths which are not dc-paths. The steps are given in Fig. 27. We may like to generate specific test cases for du-paths that are not dc-paths.

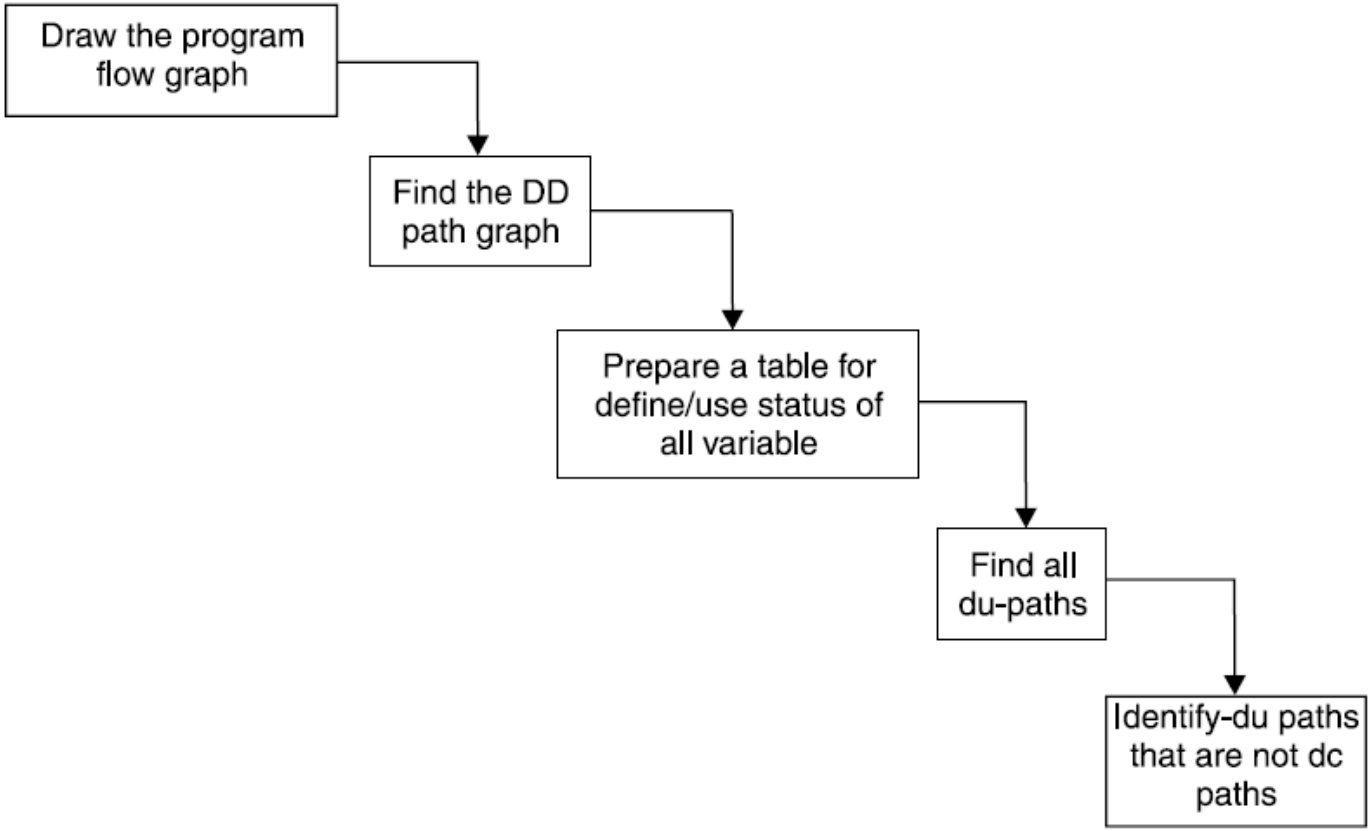
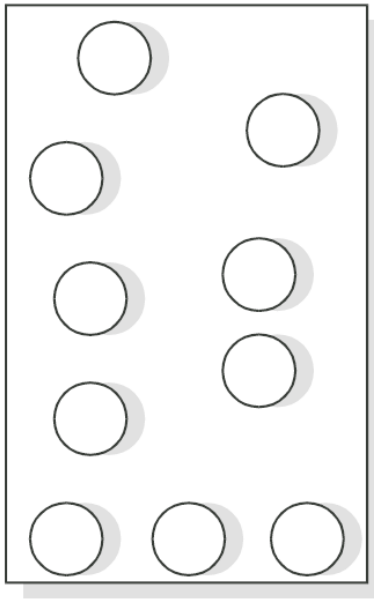


Fig. 27 : Steps for data flow testing

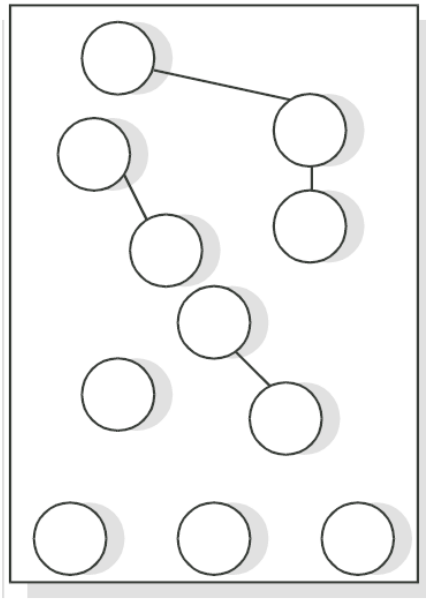
Levels of Testing

There are 3 levels of testing:

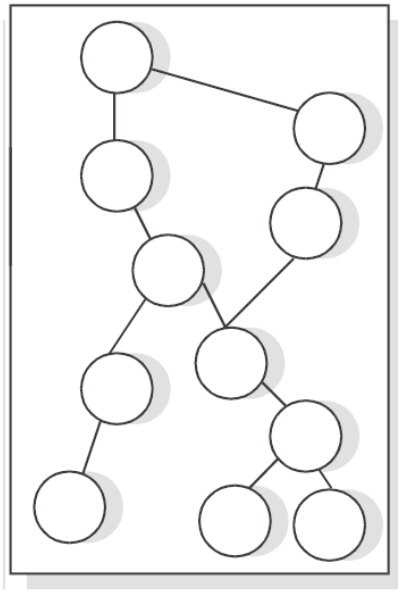
i. Unit Testing
ii. Integration Testing
iii. System Testing



UNIT TESTING



INTEGRATION TESTING



SYSTEM TESTING

Software Testing

Unit Testing

There are number of reasons in support of unit testing than testing the entire product.

1. **The size of a single module is small enough that we can locate an error fairly easily.**

2. **The module is small enough that we can attempt to test it in some demonstrably exhaustive fashion.**

3. **Confusing interactions of multiple errors in widely different parts of the software are eliminated.**

There are problems associated with testing a module in isolation. How do we run a module without anything to call it, to be called by it or, possibly, to output intermediate values obtained during execution? One approach is to construct an appropriate driver routine to call if and, simple stubs to be called by it, and to insert output statements in it.

Stubs serve to replace modules that are subordinate to (called by) the module to be tested. A stub or dummy subprogram uses the subordinate module's interface, may do minimal data manipulation, prints verification of entry, and returns.

This overhead code, called scaffolding represents effort that is import to testing, but does not appear in the delivered product as shown in Fig. 29.

Software Testing

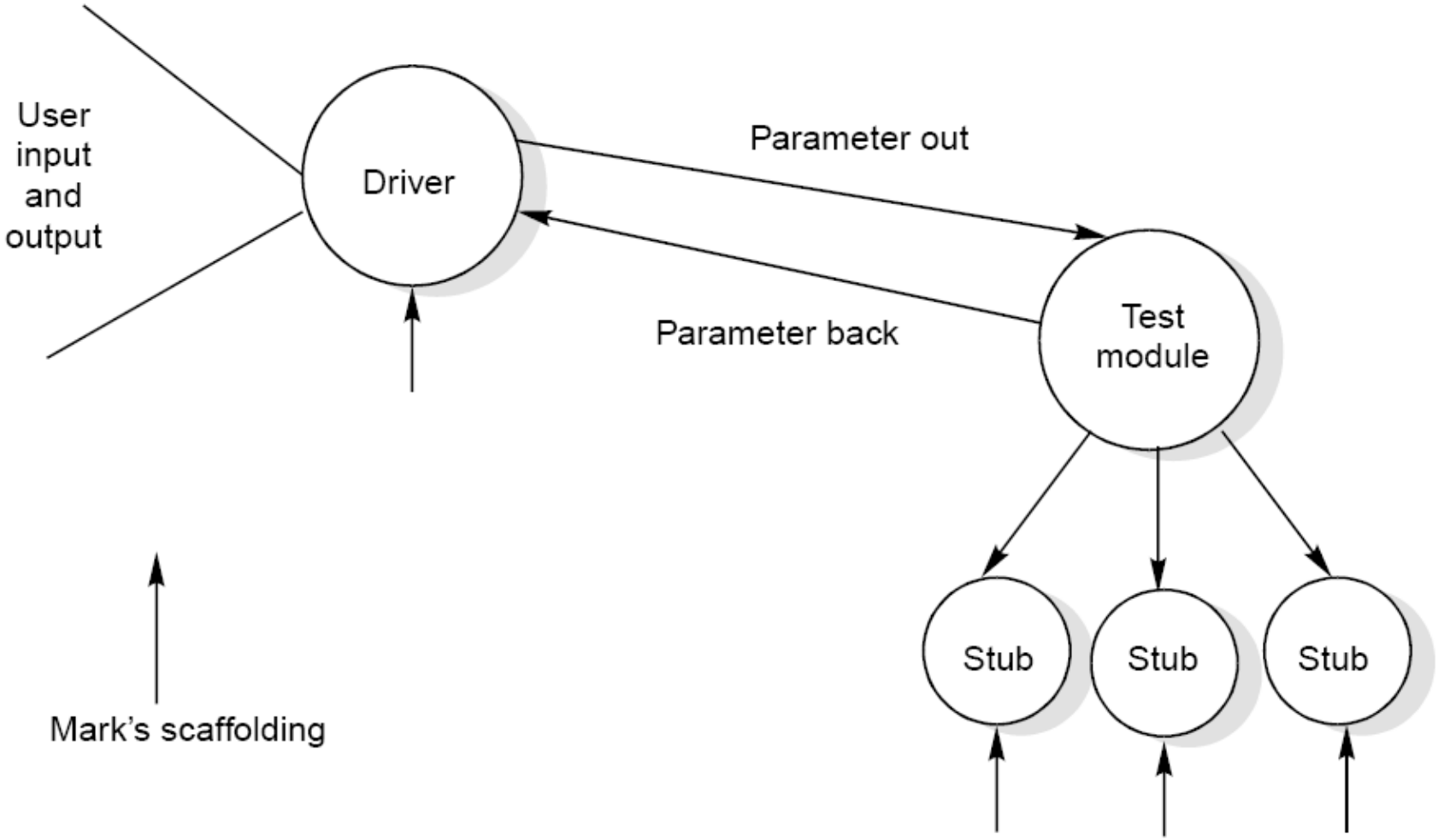
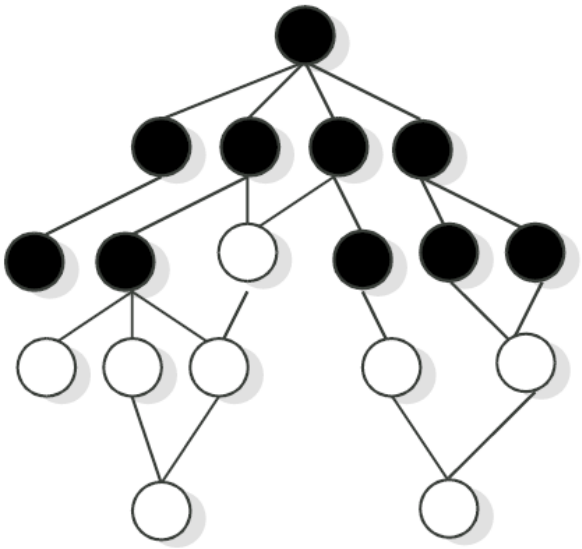


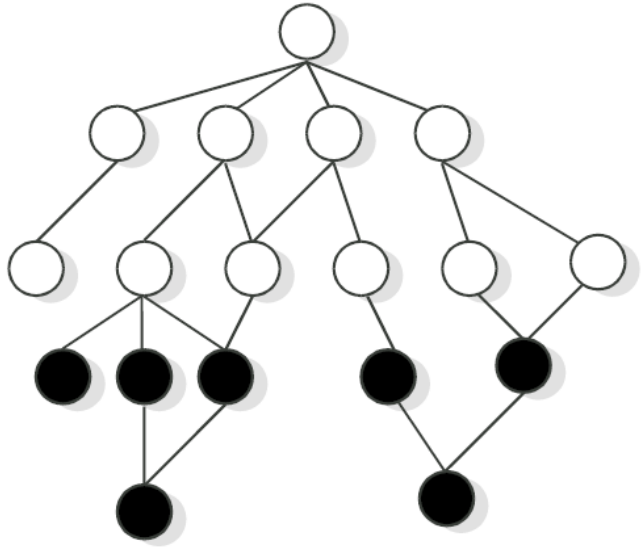
Fig. 29 : Scaffolding required testing a program unit (module)

Integration Testing

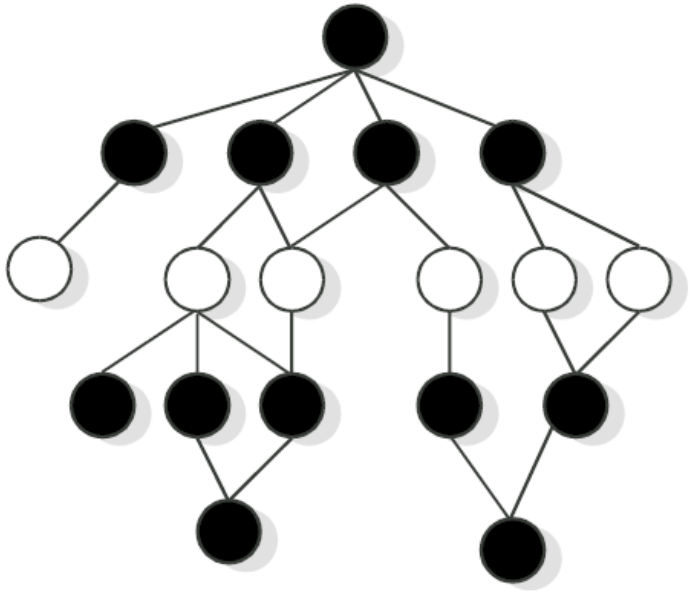
The purpose of unit testing is to determine that each independent module is correctly implemented. This gives little chance to determine that the interface between modules is also correct, and for this reason integration testing must be performed. One specific target of integration testing is the interface: whether parameters match on both sides as to type, permissible ranges, meaning and utilization.



Top-down integration



Bottom-up integration



Sandwich integration

Fig. 30 : Three different integration approaches

Software Testing

System Testing

Of the three levels of testing, the system level is closet to everyday experiences. We test many things; a used car before we buy it, an on-line cable network service before we subscribe, and so on. A common pattern in these familiar forms is that we evaluate a product in terms of our expectations; not with respect to a specification or a standard. Consequently, goal is not to find faults, but to demonstrate performance. Because of this we tend to approach system testing from a functional standpoint rather than from a structural one. Since it is so intuitively familiar, system testing in practice tends to be less formal than it might be, and is compounded by the reduced testing interval that usually remains before a delivery deadline.

Petschenik gives some guidelines for choosing test cases during system testing.

Software Testing

During system testing, we should evaluate a number of attributes of the software that are vital to the user and are listed in Fig. 31. These represent the operational correctness of the product and may be part of the software specifications.

Usable	**Is the product convenient, clear, and predictable?**
Secure	**Is access to sensitive data restricted to those with authorization?**
Compatible	**Will the product work correctly in conjunction with existing data, software, and procedures?**
Dependable	**Do adequate safeguards against failure and methods for recovery exist in the product?**
Documented	**Are manuals complete, correct, and understandable?**

Fig. 31 : Attributes of software to be tested during system testing

Software Testing

Validation Testing

- It refers to test the software as a complete product.
- This should be done after unit & integration testing.
- Alpha, beta & acceptance testing are nothing but the various ways of involving customer during testing.

Software Testing

Validation Testing

- **IEEE has developed a standard (IEEE standard 1059-1993) entitled " IEEE guide for software verification and validation " to provide specific guidance about planning and documenting the tasks required by the standard so that the customer may write an effective plan.**
- **Validation testing improves the quality of software product in terms of functional capabilities and quality attributes.**

The Art of Debugging

The goal of testing is to identify errors (bugs) in the program. The process of testing generates symptoms, and a program's failure is a clear symptom of the presence of an error. After getting a symptom, we begin to investigate the cause and place of that error. After identification of place, we examine that portion to identify the cause of the problem. This process is called debugging.

Debugging Techniques

Pressman explained few characteristics of bugs that provide some clues.

1. **"The symptom and the cause may be geographically remote. That is, the symptom may appear in one part of a program, while the cause may actually be located in other part. Highly coupled program structures may complicate this situation.**
2. **The symptom may disappear (temporarily) when another error is corrected.**

3. **The symptom may actually be caused by non errors (e.g. round off inaccuracies).**
4. **The symptom may be caused by a human error that is not easily traced.**
5. **The symptom may be a result of timing problems rather than processing problems.**
6. **It may be difficult to accurately reproduce input conditions (e.g. a real time application in which input ordering is indeterminate).**
7. **The symptom may be intermittent. This is particularly common in embedded system that couple hardware with software inextricably.**
8. **The symptom may be due to causes that are distributed across a number of tasks running on different processors".**

Software Testing

Induction approach

- Locate the pertinent data
- Organize the data
- Devise a hypothesis
- Prove the hypothesis

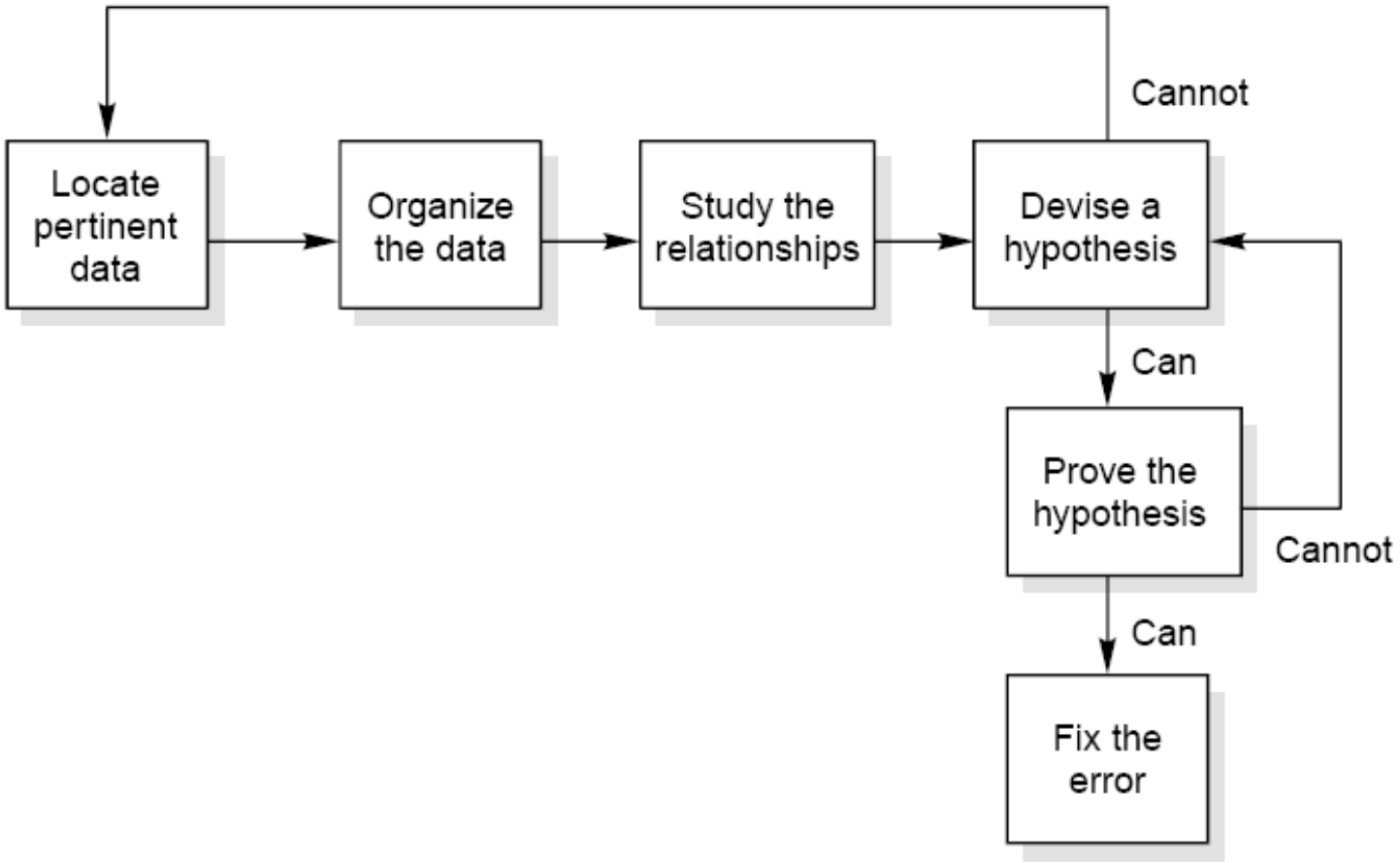


Fig. 32 : The inductive debugging process

Software Testing

Deduction approach

- Enumerate the possible causes or hypotheses
- Use the data to eliminate possible causes
- Refine the remaining hypothesis
- Prove the remaining hypothesis

Software Testing

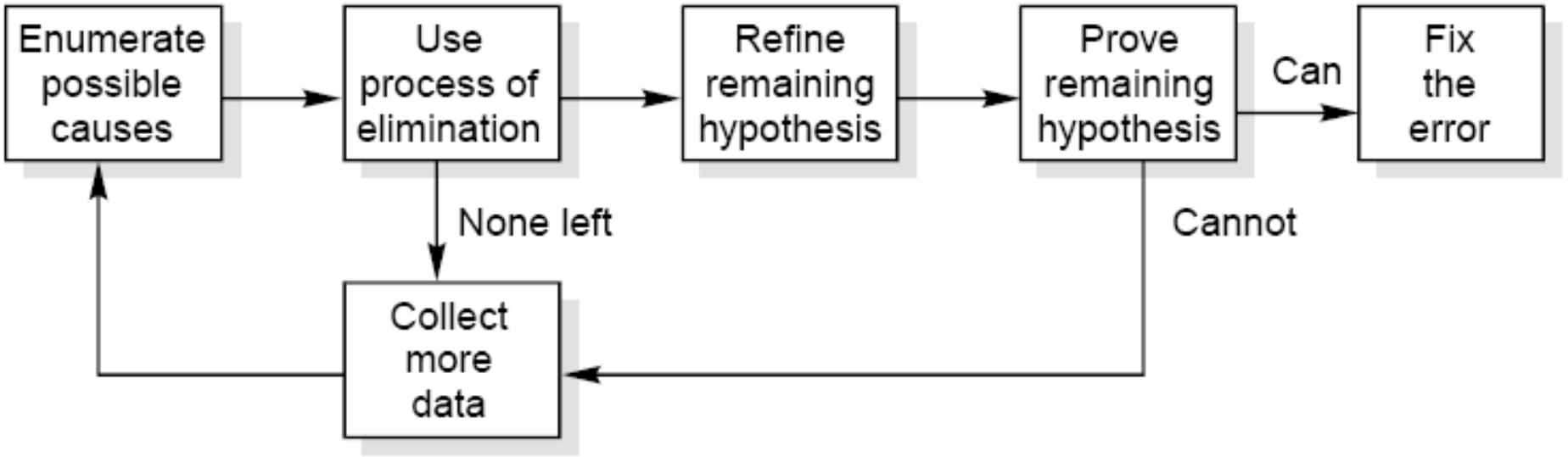


Fig. 33 : The inductive debugging process

Software testing

Lecture – 23

Software Testing

Testing Tools

One way to improve the quality & quantity of testing is to make the process as pleasant as possible for the tester. This means that tools should be as concise, powerful & natural as possible.

The two broad categories of software testing tools are :

- **Static**
- **Dynamic**

There are different types of tools available and some are listed below:

1. **Static analyzers, which examine programs systematically and automatically.**
2. **Code inspectors, who inspect programs automatically to make sure they adhere to minimum quality standards.**
3. **standards enforcers, which impose simple rules on the developer.**
4. **Coverage analysers, which measure the extent of coverage.**
5. **Output comparators, used to determine whether the output in a program is appropriate or not.**

Software Testing

6. **Test file/ data generators, used to set up test inputs.**

7. **Test harnesses, used to simplify test operations.**

8. **Test archiving systems, used to provide documentation about programs.**

Software testing

Lecture – 24

Objectives

- To discuss the distinctions between validation testing and defect testing
- To describe the principles of system and component testing
- To describe strategies for generating system test cases
- To understand the essential characteristics of tool used for test automation

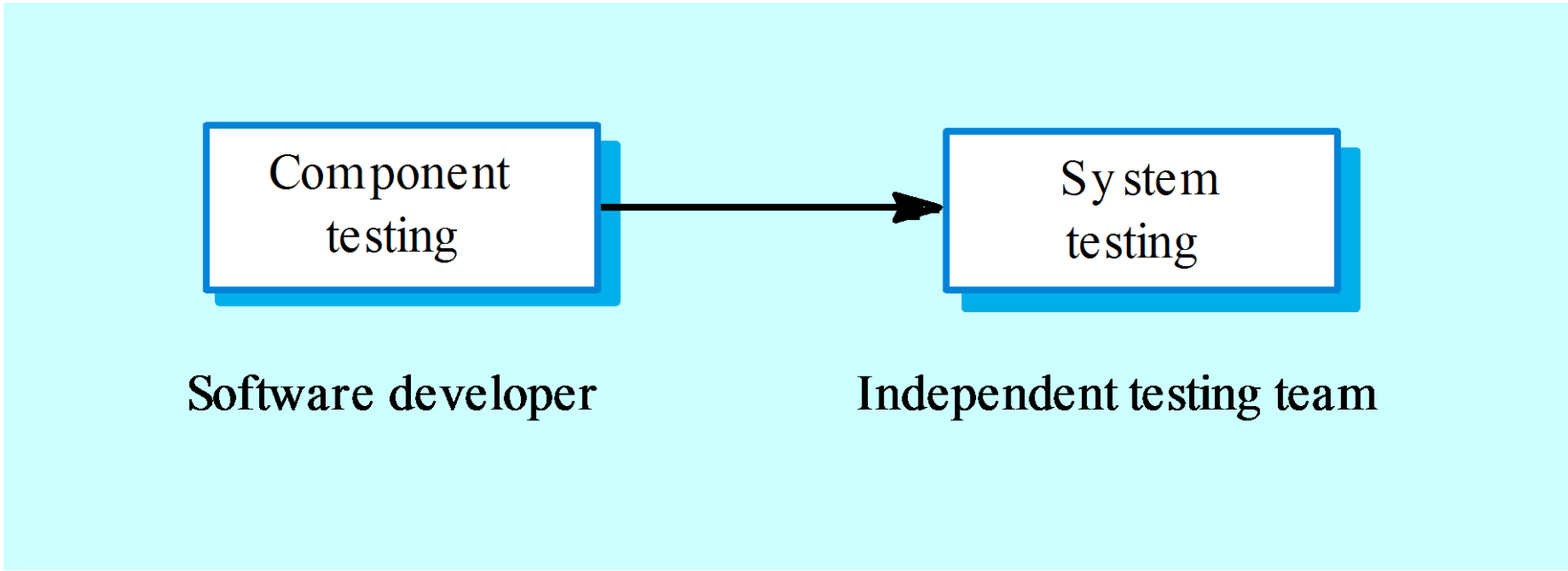
Topics covered

- System testing
- Component testing
- Test case design
- Test automation

The testing process

- Component testing
  - Testing of individual program components;
  - Usually the responsibility of the component developer (except sometimes for critical systems);
  - Tests are derived from the developer's experience.
- System testing
  - Testing of groups of components integrated to create a system or sub-system;
  - The responsibility of an independent testing team;
  - Tests are based on a system specification.

Testing phases



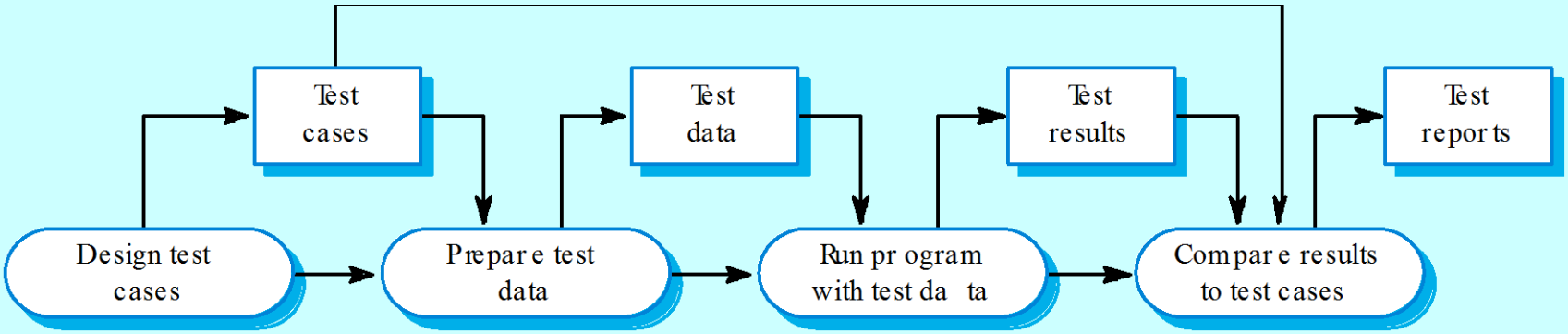
Defect testing

- The goal of defect testing is to discover defects in programs
- A *successful* defect test is a test which causes a program to behave in an anomalous way
- Tests show the presence not the absence of defects

Testing process goals

- Validation testing
  - To demonstrate to the developer and the system customer that the software meets its requirements;
  - A successful test shows that the system operates as intended.
- Defect testing
  - To discover faults or defects in the software where its behaviour is incorrect or not in conformance with its specification;
  - A successful test is a test that makes the system perform incorrectly and so exposes a defect in the system.

The software testing process



Testing policies

- Only exhaustive testing can show a program is free from defects. However, exhaustive testing is impossible,
- Testing policies define the approach to be used in selecting system tests:
  - All functions accessed through menus should be tested;
  - Combinations of functions accessed through the same menu should be tested;
  - Where user input is required, all functions must be tested with correct and incorrect input.

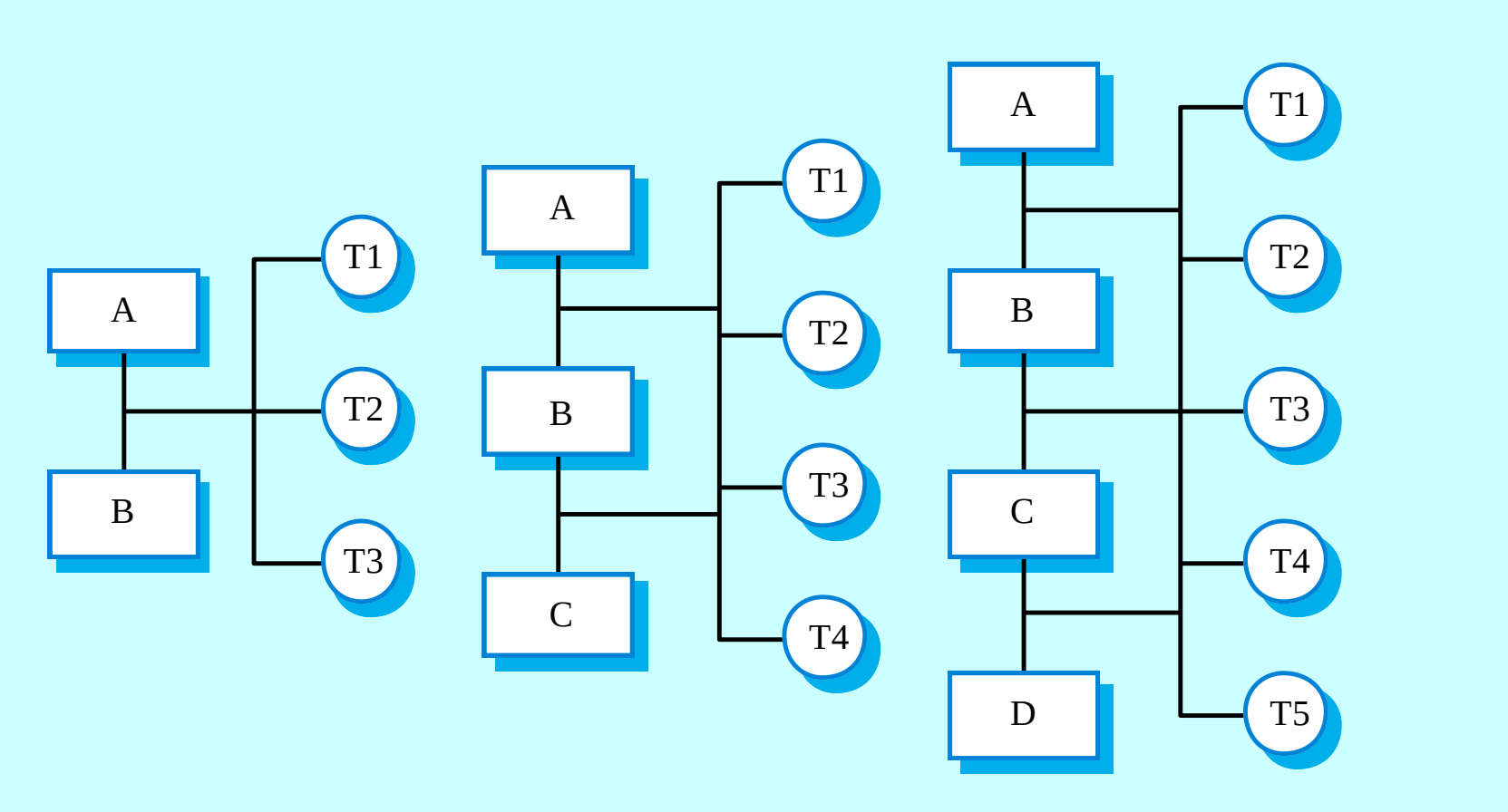
System testing

- Involves integrating components to create a system or sub-system.
- May involve testing an increment to be delivered to the customer.
- Two phases:
  - Integration testing – the test team have access to the system source code. The system is tested as components are integrated.
  - Release testing – the test team test the complete system to be delivered as a black-box.

Integration testing

- Involves building a system from its components and testing it for problems that arise from component interactions.
- Top-down integration
  - Develop the skeleton of the system and populate it with components.
- Bottom-up integration
  - Integrate infrastructure components then add functional components.
- To simplify error localisation, systems should be incrementally integrated.

Incremental integration testing



Test sequence 1 Test sequence 2 Test sequence 3

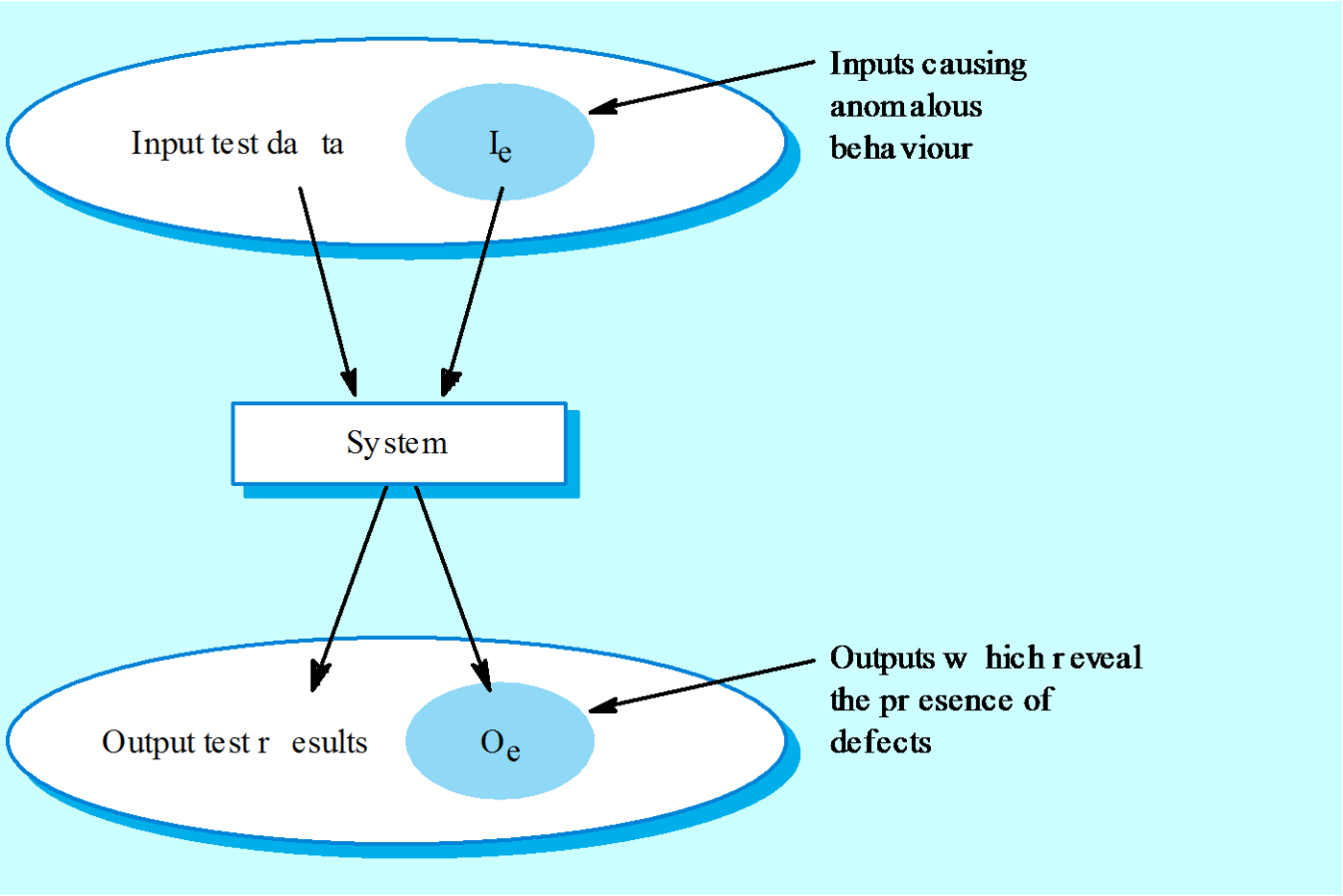
Testing approaches

- Architectural validation
  - Top-down integration testing is better at discovering errors in the system architecture.
- System demonstration
  - Top-down integration testing allows a limited demonstration at an early stage in the development.
- Test implementation
  - Often easier with bottom-up integration testing.
- Test observation
  - Problems with both approaches. Extra code may be required to observe tests.

Release testing

- The process of testing a release of a system that will be distributed to customers.
- Primary goal is to increase the supplier's confidence that the system meets its requirements.
- Release testing is usually black-box or functional testing
  - Based on the system specification only;
  - Testers do not have knowledge of the system implementation.

Black-box testing



Testing guidelines

- Testing guidelines are hints for the testing team to help them choose tests that will reveal defects in the system
  - Choose inputs that force the system to generate all error messages;
  - Design inputs that cause buffers to overflow;
  - Repeat the same input or input series several times;
  - Force invalid outputs to be generated;
  - Force computation results to be too large or too small.

Testing scenario

A student in Scotland is studying American History and has been asked to write a paper on ÔFrontier mentality in the American West from 1840 to 1880Õ. To do this, she needs to find sources from a range of libraries. She logs on to the LIBSYS system and uses the search facility to discover if she can access original documents from that time. She discovers sources in various US university libraries and downloads copies of some of these. However, for one document, she needs to have confirmation from her university that she is a genuine student and that use is for non-commercial purposes. The student then uses the facility in LIBSYS that can request such permission and registers her request. If granted, the document will be downloaded to the registered libraryÕs server and printed for her. She receives a message from LIBSYS telling her that she will receive an e-mail message when the printed document is available for collection.

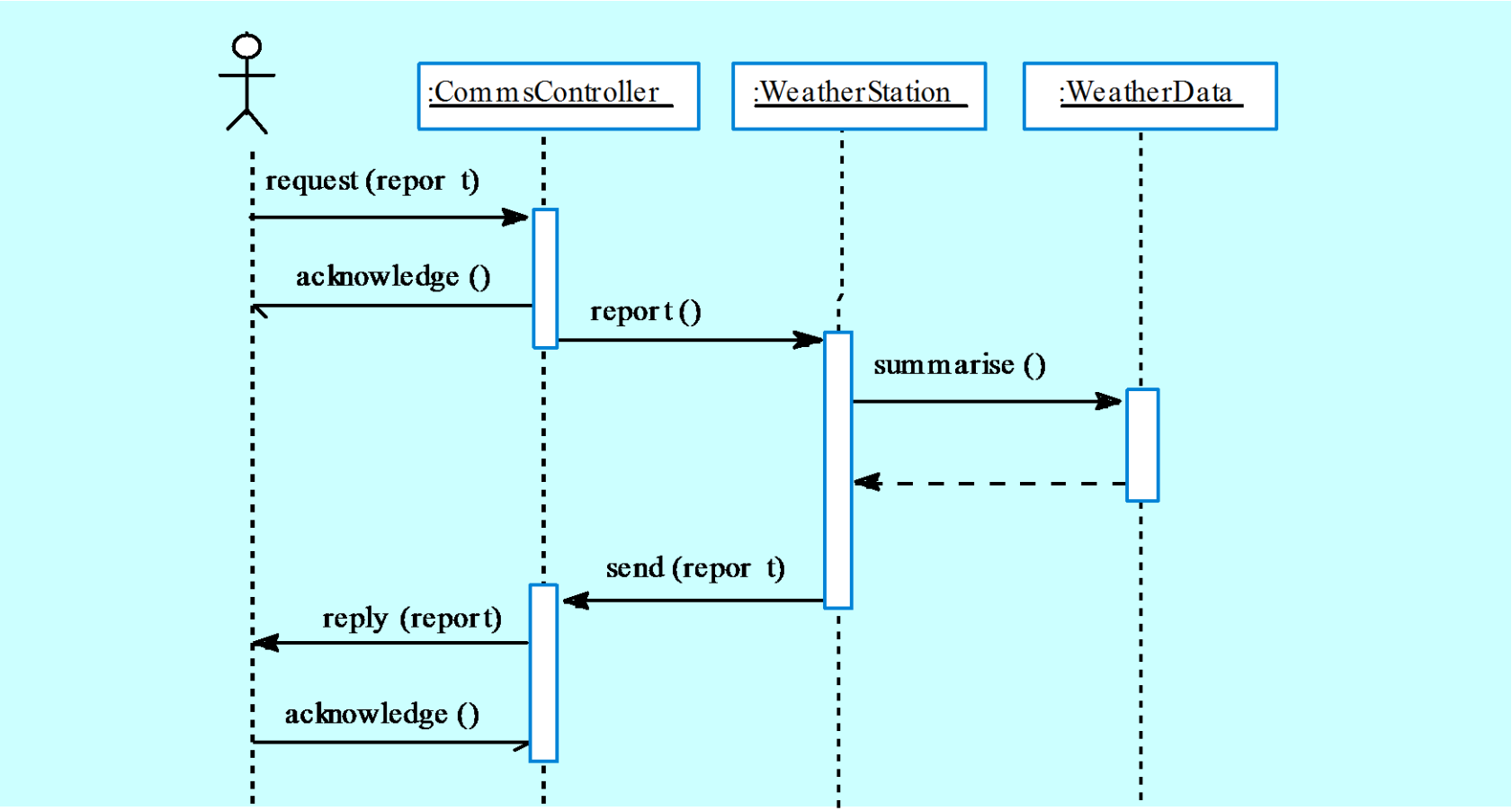
System tests

1. Test the login mechanism using correct and incorrect logins to check that valid users are accepted and invalid users are rejected.
2. Test the search facility using different queries against known sources to check that the search mechanism is actually finding documents.
3. Test the system presentation facility to check that information about documents is displayed properly.
4. Test the mechanism to request permission for downloading.
5. Test the e-mail response indicating that the downloaded document is available.

Use cases

- Use cases can be a basis for deriving the tests for a system. They help identify operations to be tested and help design the required test cases.
- From an associated sequence diagram, the inputs and outputs to be created for the tests can be identified.

Collect weather data sequence chart



Performance testing

- Part of release testing may involve testing the emergent properties of a system, such as performance and reliability.
- Performance tests usually involve planning a series of tests where the load is steadily increased until the system performance becomes unacceptable.

Stress testing

- Exercises the system beyond its maximum design load. Stressing the system often causes defects to come to light.
- Stressing the system test failure behaviour.. Systems should not fail catastrophically. Stress testing checks for unacceptable loss of service or data.
- Stress testing is particularly relevant to distributed systems that can exhibit severe degradation as a network becomes overloaded.

Component testing

- Component or unit testing is the process of testing individual components in isolation.
- It is a defect testing process.
- Components may be:
  - Individual functions or methods within an object;
  - Object classes with several attributes and methods;
  - Composite components with defined interfaces used to access their functionality.

Object class testing

- Complete test coverage of a class involves
  - Testing all operations associated with an object;
  - Setting and interrogating all object attributes;
  - Exercising the object in all possible states.
- Inheritance makes it more difficult to design object class tests as the information to be tested is not localised.

Test case design

- Involves designing the test cases (inputs and outputs) used to test the system.
- The goal of test case design is to create a set of tests that are effective in validation and defect testing.
- Design approaches:
  - Requirements-based testing;
  - Partition testing;
  - Structural testing.

Requirements based testing

- A general principle of requirements engineering is that requirements should be testable.
- Requirements-based testing is a validation testing technique where you consider each requirement and derive a set of tests for that requirement.

LIBSYS requirements

The user shall be able to search either all of the initial set of databases or select a subset from it.

The system shall provide appropriate viewers for the user to read documents in the document store.

Every order shall be allocated a unique identifier (ORDER_ID) that the user shall be able to copy to the accountÕs permanent storage area.

LIBSYS tests

- Initiate user search for searches for items that are known to be present and known not to be present, where the set of databases includes 1 database.
- Initiate user searches for items that are known to be present and known not to be present, where the set of databases includes 2 databases
- Initiate user searches for items that are known to be present and known not to be present where the set of databases includes more than 2 databases.
- Select one database from the set of databases and initiate user searches for items that are known to be present and known not to be present.
- Select more than one database from the set of databases and initiate searches for items that are known to be present and known not to be present.

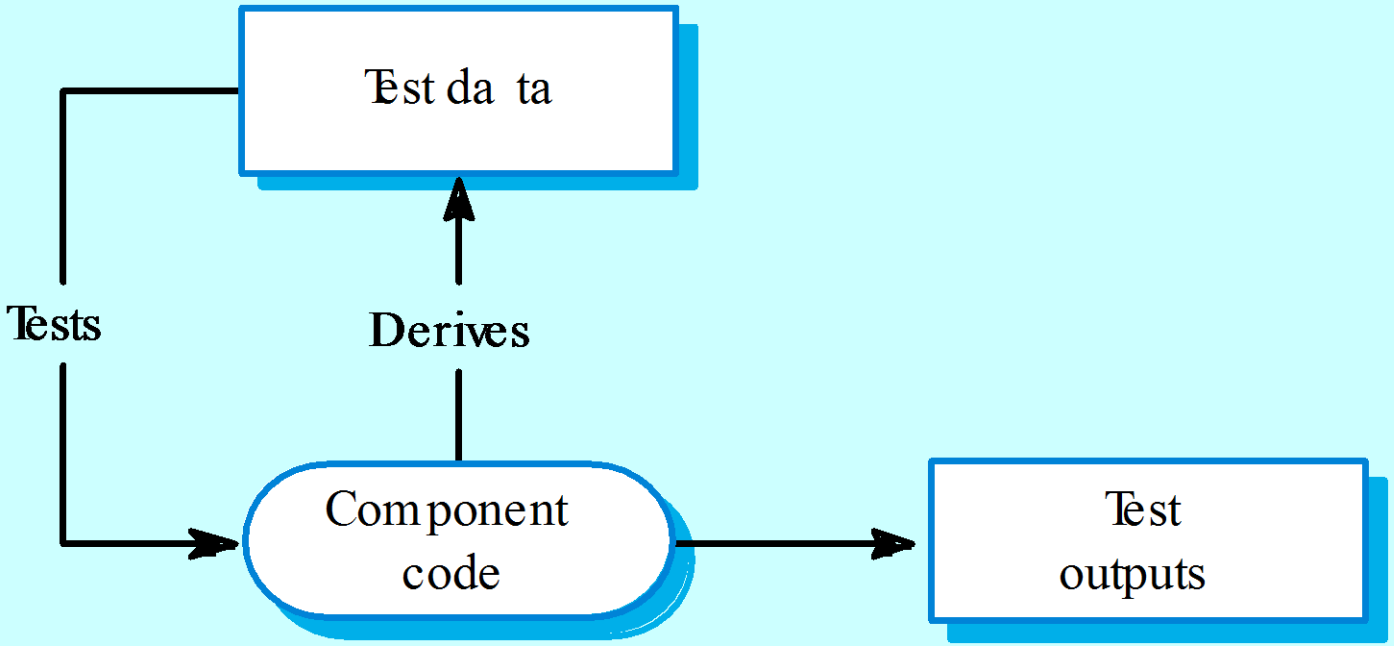
Software testing

Lecture – 25

Structural testing

- Sometime called white-box testing.
- Derivation of test cases according to program structure. Knowledge of the program is used to identify additional test cases.
- Objective is to exercise all program statements (not all path combinations).

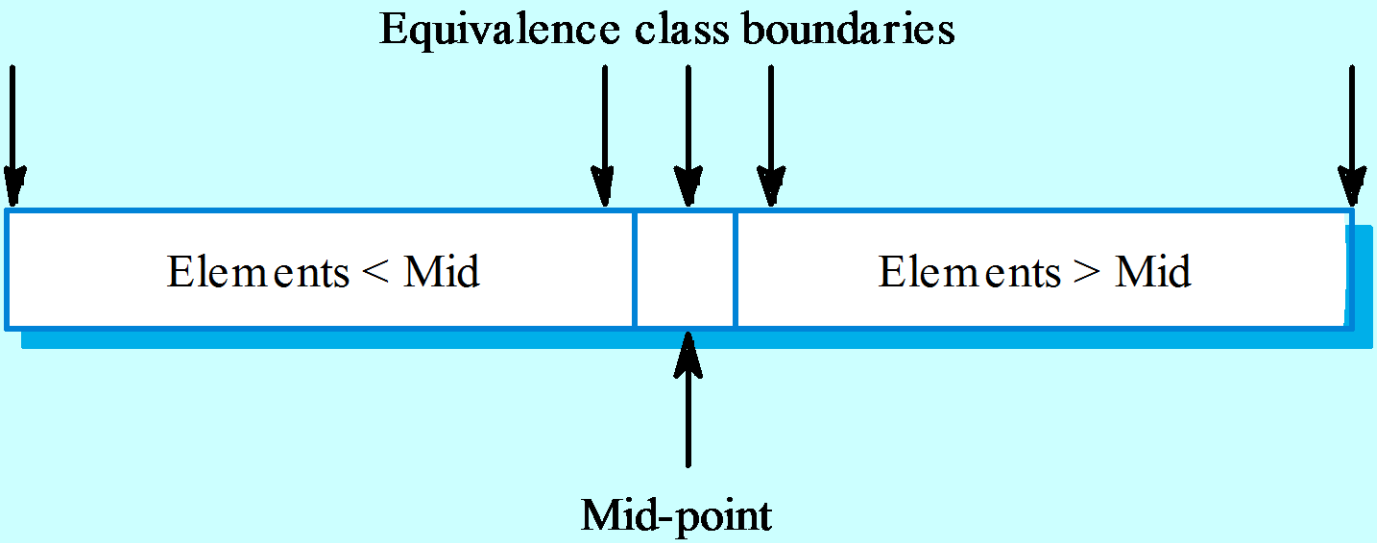
Structural testing



Binary search – equiv. partitions

- Pre-conditions satisfied, key element in array.
- Pre-conditions satisfied, key element not in array.
- Pre-conditions unsatisfied, key element in array.
- Pre-conditions unsatisfied, key element not in array.
- Input array has a single value.
- Input array has an even number of values.
- Input array has an odd number of values.

Binary search equiv. partitions



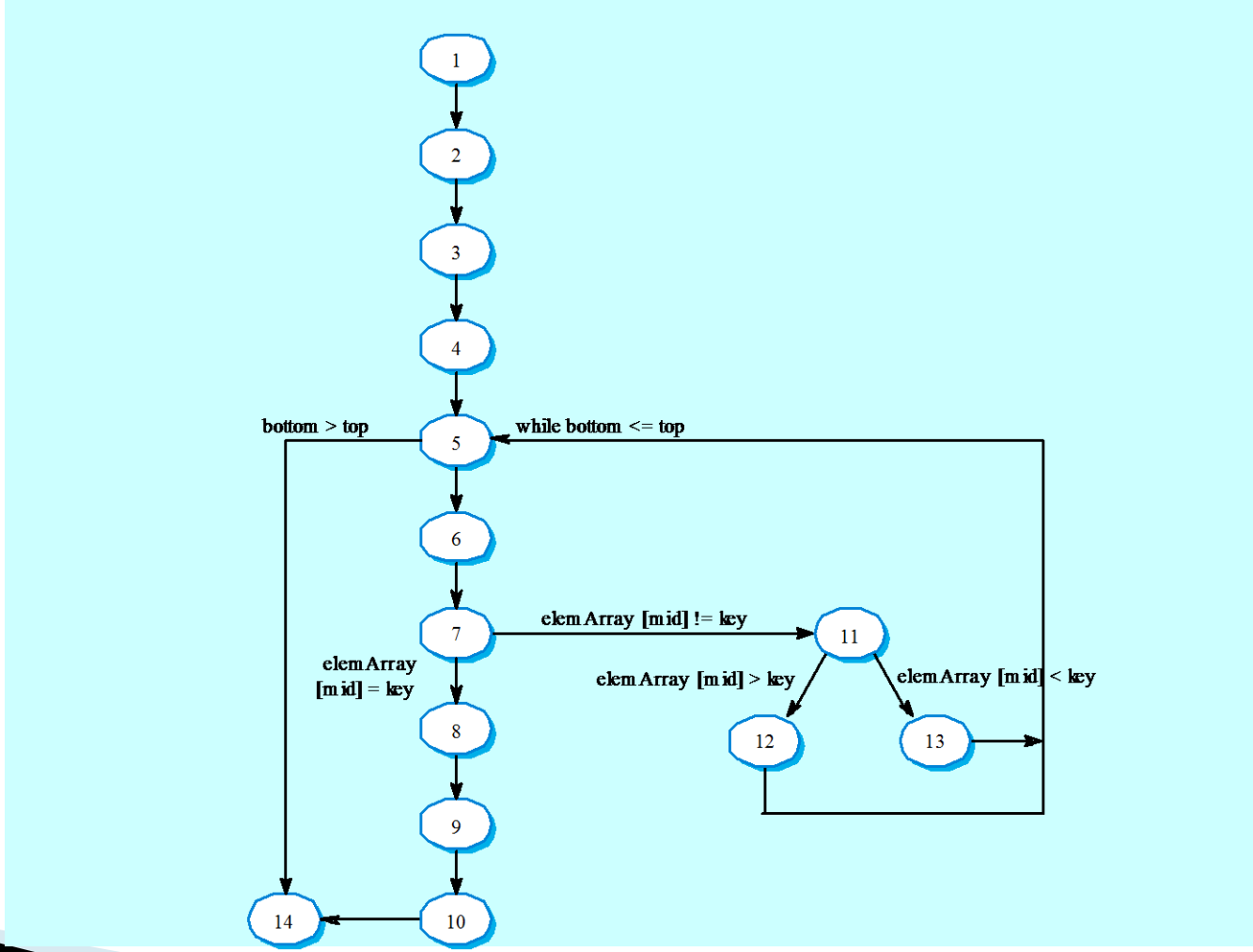
Binary search – test cases

Input array (T)	**Key** (Key)	**Output** (Found, L)
17	17	true, 1
17	0	false, ??
17, 21, 23, 29	17	true, 1
9, 16, 18, 30, 31, 41, 45	45	true, 7
17, 18, 21, 23, 29, 38, 41	23	true, 4
17, 18, 21, 23, 29, 33, 38	21	true, 3
12, 18, 21, 23, 32	23	true, 4
21, 23, 29, 33, 38	25	false, ??

Path testing

- The objective of path testing is to ensure that the set of test cases is such that each path through the program is executed at least once.
- The starting point for path testing is a program flow graph that shows nodes representing program decisions and arcs representing the flow of control.
- Statements with conditions are therefore nodes in the flow graph.

Binary search flow graph



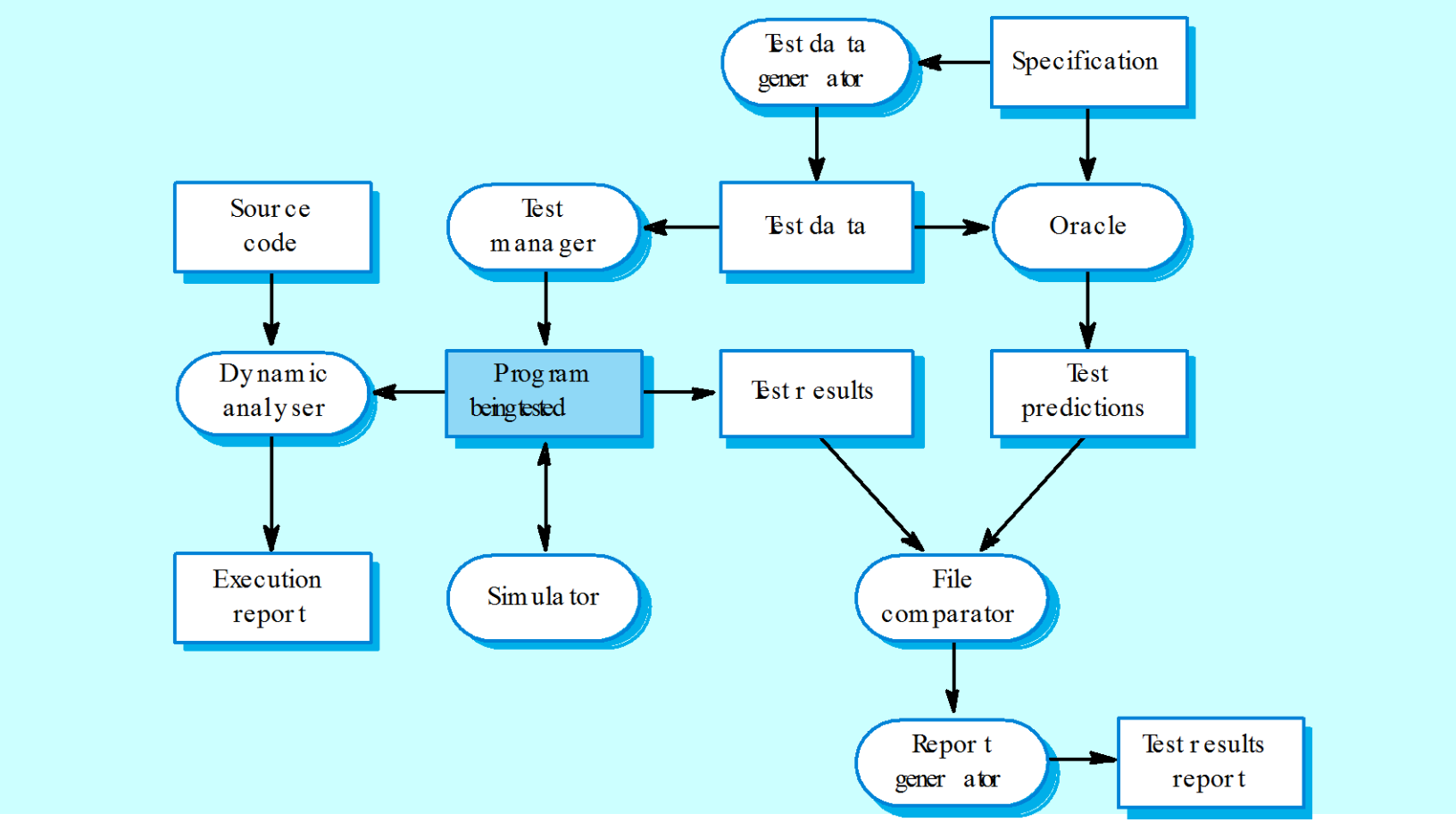
Independent paths

- 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 14
- 1, 2, 3, 4, 5, 14
- 1, 2, 3, 4, 5, 6, 7, 11, 12, 5, …
- 1, 2, 3, 4, 6, 7, 2, 11, 13, 5, …
- Test cases should be derived so that all of these paths are executed
- A dynamic program analyser may be used to check that paths have been executed

Test automation

- Testing is an expensive process phase. Testing workbenches provide a range of tools to reduce the time required and total testing costs.
- Systems such as Junit support the automatic execution of tests.
- Most testing workbenches are open systems because testing needs are organisation-specific.
- They are sometimes difficult to integrate with closed design and analysis workbenches.

A testing workbench



Testing workbench adaptation

- Scripts may be developed for user interface simulators and patterns for test data generators.
- Test outputs may have to be prepared manually for comparison.
- Special-purpose file comparators may be developed.

Key points

- Testing can show the presence of faults in a system; it cannot prove there are no remaining faults.
- Component developers are responsible for component testing; system testing is the responsibility of a separate team.
- Integration testing is testing increments of the system; release testing involves testing a system to be released to a customer.
- Use experience and guidelines to design test cases in defect testing.

Key points

- Interface testing is designed to discover defects in the interfaces of composite components.
- Equivalence partitioning is a way of discovering test cases – all cases in a partition should behave in the same way.
- Structural analysis relies on analysing a program and deriving tests from this analysis.
- Test automation reduces testing costs by supporting the test process with a range of software tools.

Verification and Validation

Lecture – 26

Objectives

- To introduce software verification and validation and to discuss the distinction between them
- To describe the program inspection process and its role in V & V
- To explain static analysis as a verification technique
- To describe the Cleanroom software development process

Topics covered

- Verification and validation planning
- Software inspections
- Automated static analysis
- Cleanroom software development

Verification vs validation

- Verification:
  "Are we building the product right".
- The software should conform to its specification.
- Validation:
  "Are we building the right product".
- The software should do what the user really requires.

The V & V process

- Is a whole life-cycle process - V & V must be applied at each stage in the software process.
- Has two principal objectives
  - The discovery of defects in a system;
  - The assessment of whether or not the system is useful and useable in an operational situation.

V& V goals

- Verification and validation should establish confidence that the software is fit for purpose.
- This does NOT mean completely free of defects.
- Rather, it must be good enough for its intended use and the type of use will determine the degree of confidence that is needed.

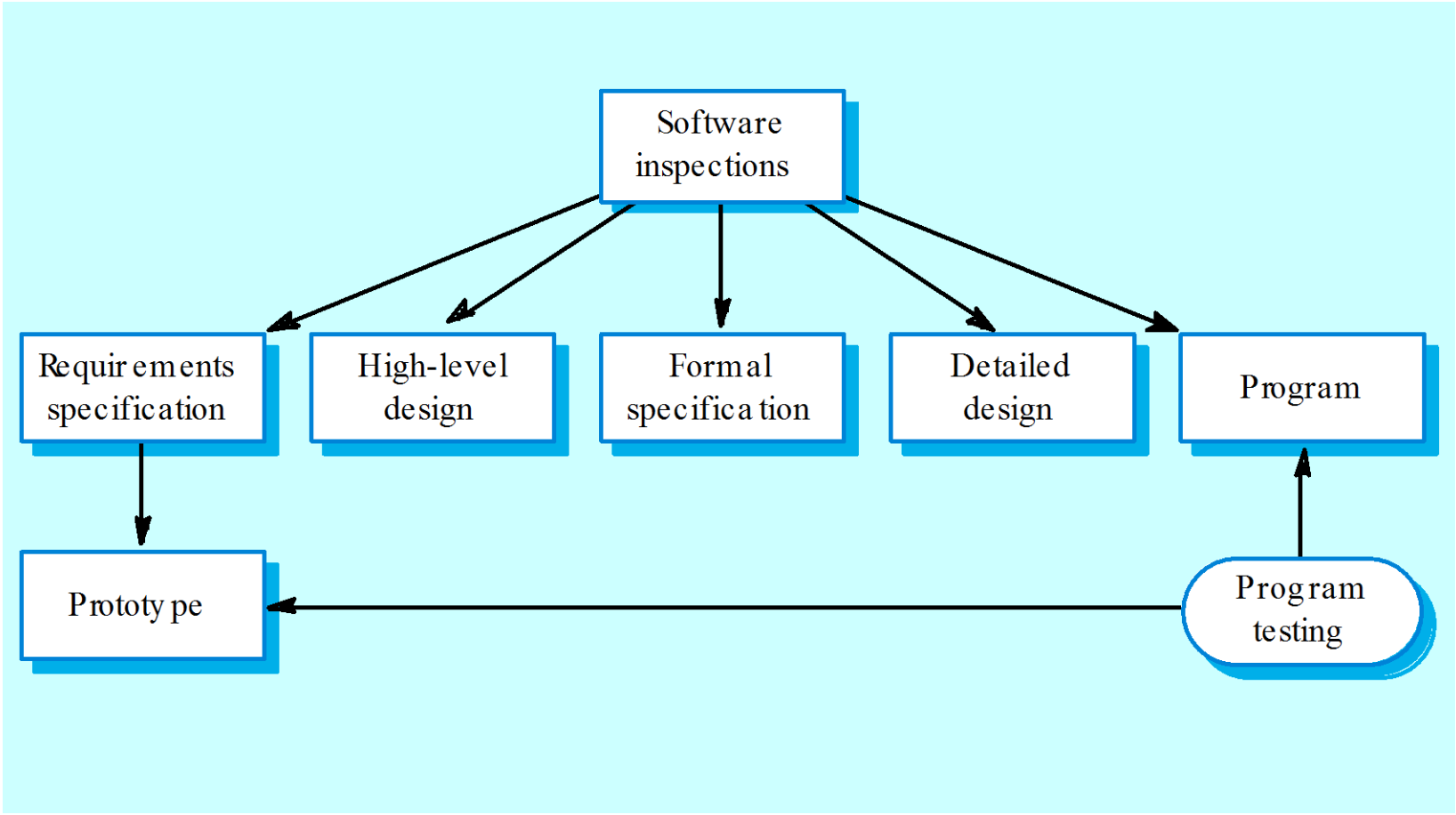
V & V confidence

- Depends on system's purpose, user expectations and marketing environment
  - Software function
    - The level of confidence depends on how critical the software is to an organisation.
  - User expectations
    - Users may have low expectations of certain kinds of software.
  - Marketing environment
    - Getting a product to market early may be more important than finding defects in the program.

Static and dynamic verification

- Software inspections. Concerned with analysis of
  the static system representation to discover problems (static verification)
  - May be supplement by tool-based document and code analysis
- Software testing. Concerned with exercising and
  observing product behaviour (dynamic verification)
  - The system is executed with test data and its operational behaviour is observed

Static and dynamic V&V



Program testing

- Can reveal the presence of errors NOT their absence.
- The only validation technique for non-functional requirements as the software has to be executed to see how it behaves.
- Should be used in conjunction with static verification to provide full V&V coverage.

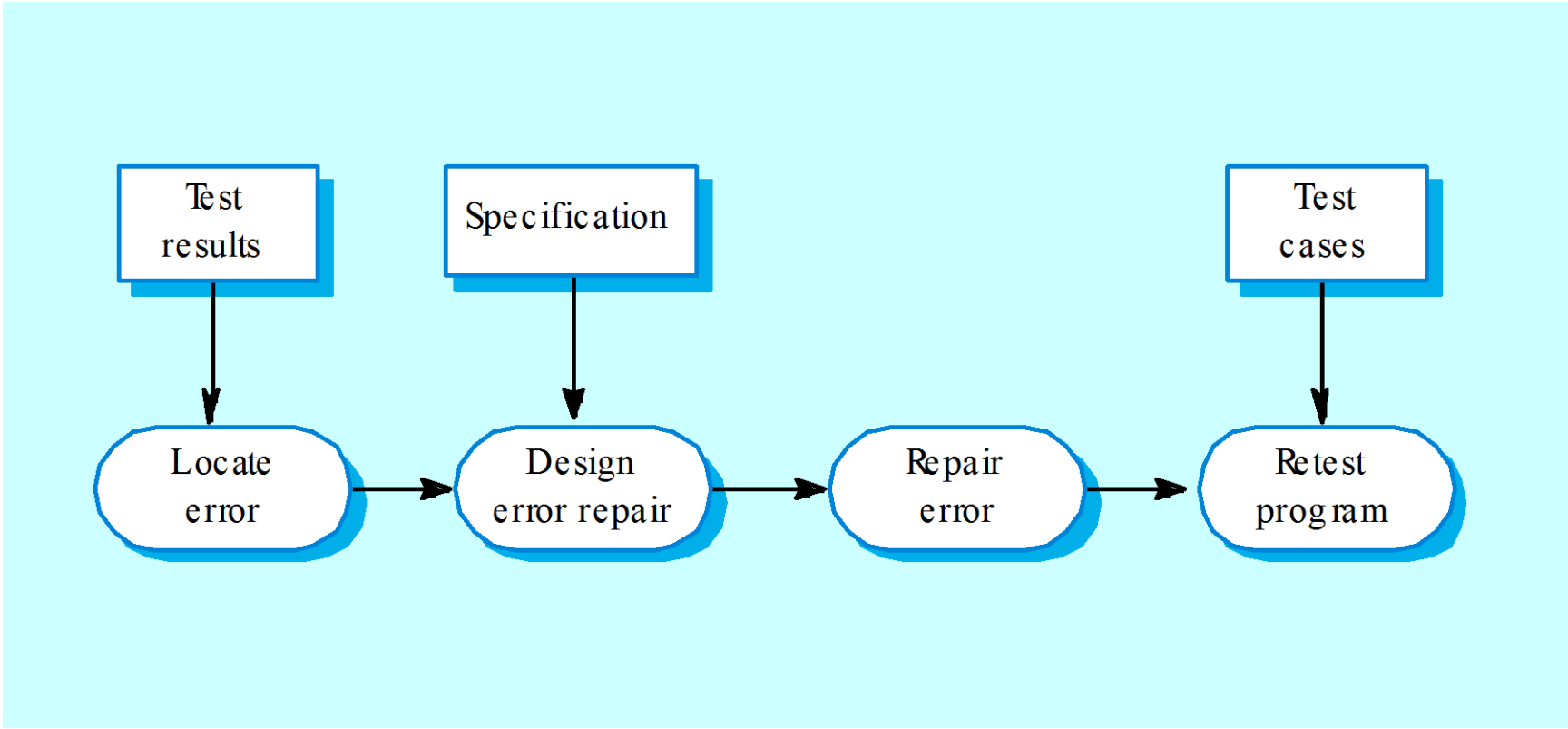
Types of testing

- Defect testing
  - Tests designed to discover system defects.
  - A successful defect test is one which reveals the presence of defects in a system.
  - Covered in Chapter 23
- Validation testing
  - Intended to show that the software meets its requirements.
  - A successful test is one that shows that a requirements has been properly implemented.

Testing and debugging

- Defect testing and debugging are distinct processes.
- Verification and validation is concerned with establishing the existence of defects in a program.
- Debugging is concerned with locating and repairing these errors.
- Debugging involves formulating a hypothesis about program behaviour then testing these hypotheses to find the system error.

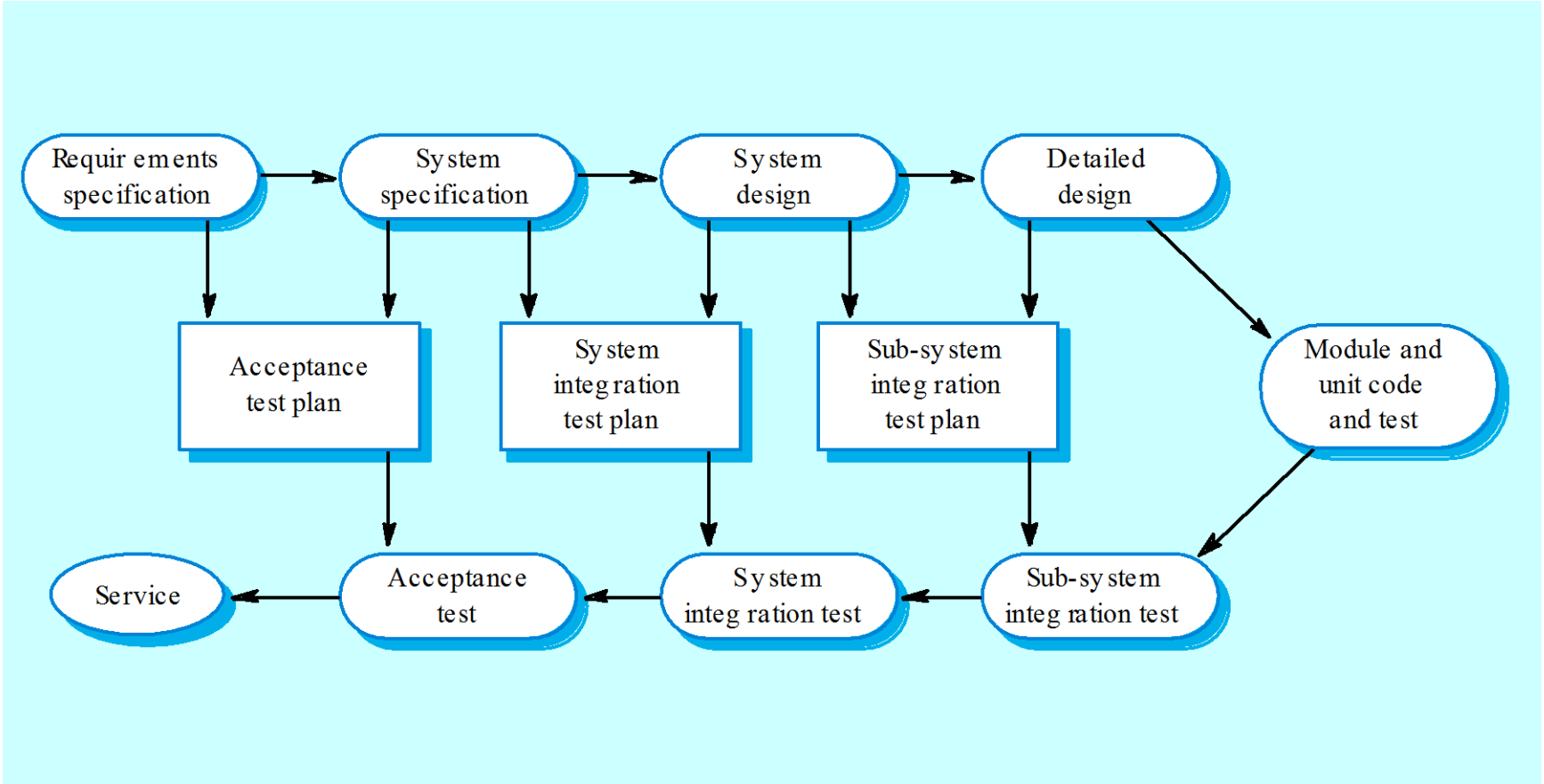
The debugging process



V & V planning

- Careful planning is required to get the most out of testing and inspection processes.
- Planning should start early in the development process.
- The plan should identify the balance between static verification and testing.
- Test planning is about defining standards for the testing process rather than describing product tests.

The V-model of development



The structure of a software test plan

- The testing process.
- Requirements traceability.
- Tested items.
- Testing schedule.
- Test recording procedures.
- Hardware and software requirements.
- Constraints.

The software test plan

The testing process
A description of the major phases of the testing process. These might be as described earlier in this chapter.

Requirements traceability
Users are most interested in the system meeting its requirements and testing should be planned so that all requirements are individually tested.

Tested items
The products of the software process that are to be tested should be specified.

Testing schedule
An overall testing schedule and resource allocation for this schedule. This, obviously, is linked to the more general project development schedule.

Test recording procedures
It is not enough simply to run tests. The results of the tests must be systematically recorded. It must be possible to audit the testing process to check that it been carried out correctly.

Hardware and software requirements
This section should set out software tools required and estimated hardware utilisation.

Constraints
Constraints affecting the testing process such as staff shortages should be anticipated in this section.

Software Maintenance

Lecture – 27

Software Maintenance

What is Software Maintenance?

Software Maintenance is a very broad activity that includes error corrections, enhancements of capabilities, deletion of obsolete capabilities, and optimization.

Software Maintenance

Categories of Maintenance

- **Corrective maintenance**

This refer to modifications initiated by defects in the software.

- **Adaptive maintenance**

It includes modifying the software to match changes in the ever changing environment.

- **Perfective maintenance**

It means improving processing efficiency or performance, or restructuring the software to improve changeability. This may include enhancement of existing system functionality, improvement in computational efficiency etc.

Software Maintenance

- **Other types of maintenance**

There are long term effects of corrective, adaptive and perfective changes. This leads to increase in the complexity of the software, which reflect deteriorating structure. The work is required to be done to maintain it or to reduce it, if possible. This work may be named as preventive maintenance.

Software Maintenance

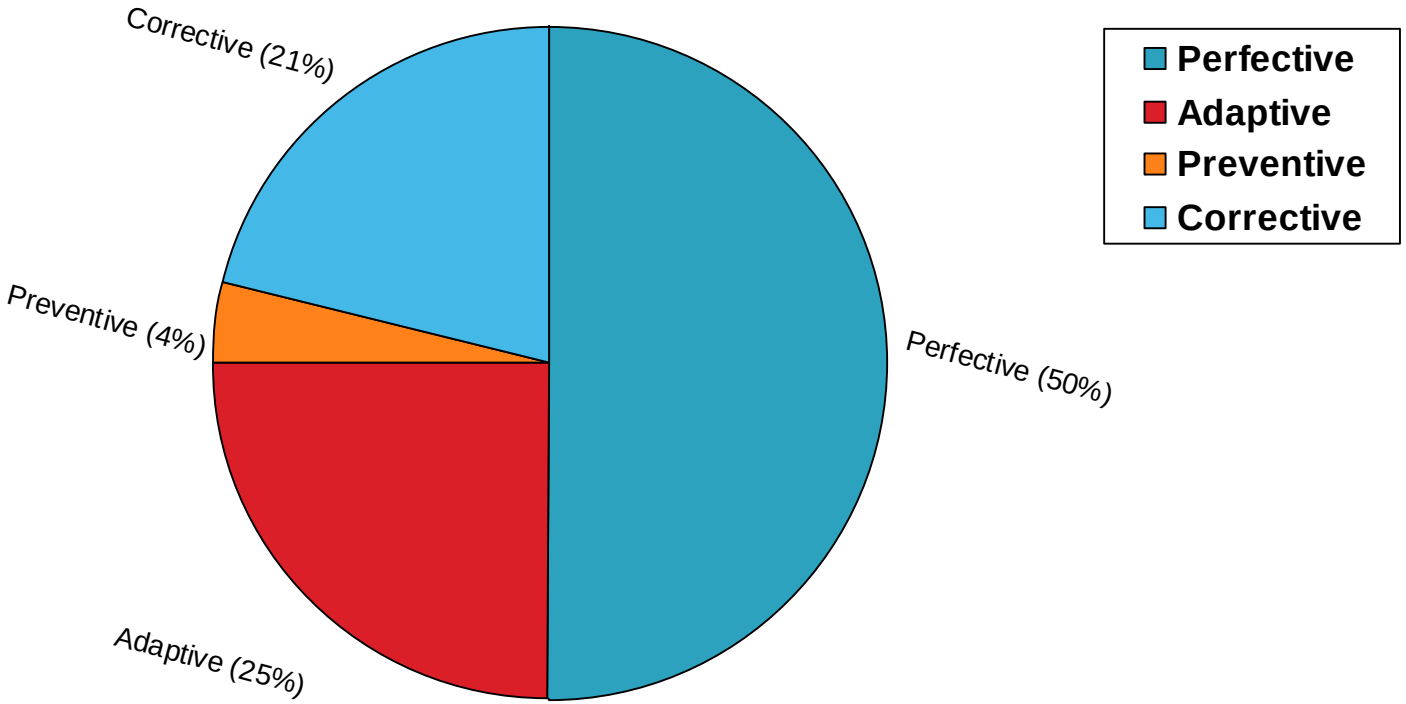


Fig. 1: Distribution of maintenance effort

Software Maintenance

Problems During Maintenance

- **Often the program is written by another person or group of persons.**
- **Often the program is changed by person who did not understand it clearly.**
- **Program listings are not structured.**
- **High staff turnover.**
- **Information gap.**
- **Systems are not designed for change.**

Maintenance is Manageable

A common misconception about maintenance is that it is not manageable.

Report of survey conducted by Lientz & Swanson gives some interesting observations:

1	Emergency debugging	12.4%
2	Routine debugging	9.3%
3	Data environment adaptation	17.3%
4	Changes in hardware and OS	6.2%
5	Enhancements for users	41.8%
6	Documentation Improvement	5.5%
7	Code efficiency improvement	4.0%
8	Others	3.5%

Table 1: Distribution of maintenance effort

Software Maintenance

Kinds of maintenance requests

1	New reports	40.8%
2	Add data in existing reports	27.1%
3	Reformed reports	10%
4	Condense reports	5.6%
5	Consolidate reports	6.4%
6	Others	10.1%

Table 2: Kinds of maintenance requests

Software Maintenance

Potential Solutions to Maintenance Problems

- **Budget and effort reallocation**
- **Complete replacement of the system**
- **Maintenance of existing system**

Software Maintenance

The Maintenance Process

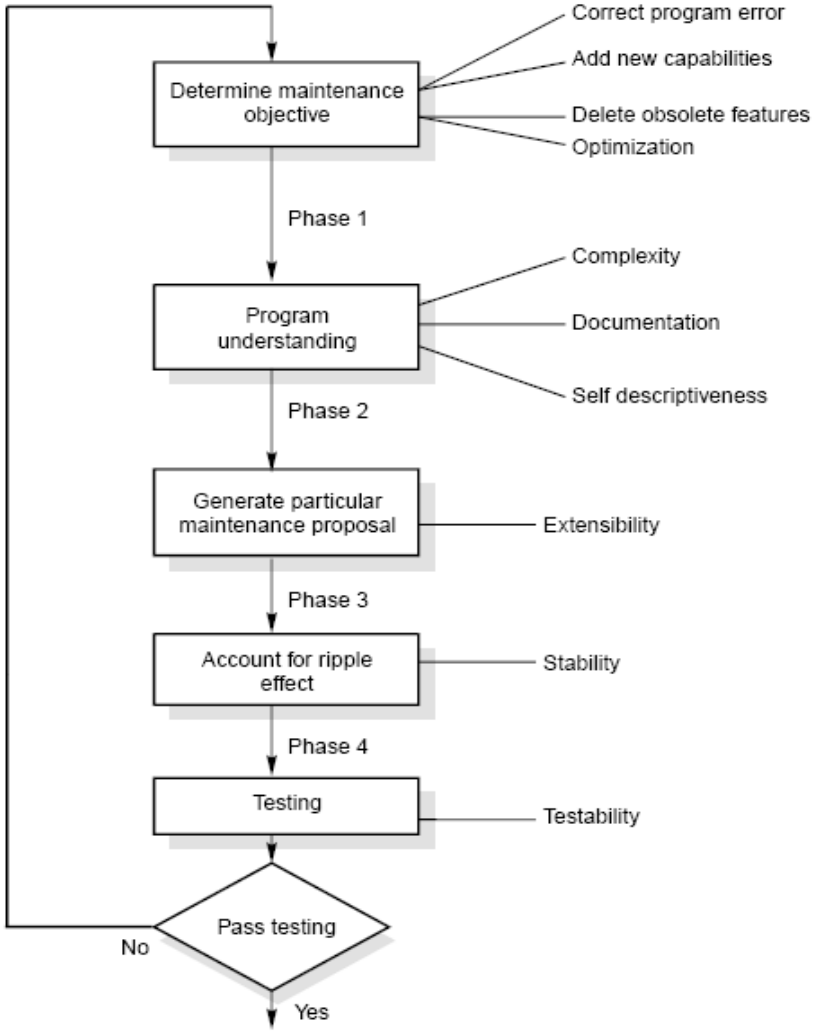


Fig. 2: The software maintenance process

Software Maintenance

- **Program Understanding**

The first phase consists of analyzing the program in order to understand.

- **Generating Particular Maintenance Proposal**

The second phase consists of generating a particular maintenance proposal to accomplish the implementation of the maintenance objective.

- **Ripple (flow) Effect**

The third phase consists of accounting for all of the ripple effect as a consequence of program modifications.

Software Maintenance

- **Modified Program Testing**

The fourth phase consists of testing the modified program to ensure that the modified program has at least the same reliability level as before.

- **Maintainability**

Each of these four phases and their associated software quality attributes are critical to the maintenance process. All of these factors must be combined to form maintainability.

Software Maintenance

Regression Testing

Regression testing is the process of retesting the modified parts of the software and ensuring that no new errors have been introduced into previously test code.

"Regression testing tests both the modified code and other parts of the program that may be affected by the program change. It serves many purposes :

- **increase confidence in the correctness of the modified program**
- **locate errors in the modified program**
- **preserve the quality and reliability of software**
- **ensure the software's continued operation**

Software Maintenance

- **Development Testing Versus Regression Testing**

Sr. No.	Development testing	Regression testing
1.	We create test suites and test plans	We can make use of existing test suite and test plans
2.	We test all software components	We retest affected components that have been modified by modifications.
3.	Budget gives time for testing	Budget often does not give time for regression testing.
4.	We perform testing just once on a software product	We perform regression testing many times over the life of the software product.
5.	Performed under the pressure of release date of the software	Performed in crisis situations, under greater time constraints.

Software Maintenance

- **Regression Test Selection**

Regression testing is very expensive activity and consumes significant amount of effort / cost. Many techniques are available to reduce this effort/ cost.

1. **Reuse the whole test suite**

2. **Reuse the existing test suite, but to apply a regression test selection technique to select an appropriate subset of the test suite to be run.**

Software Maintenance

Fragment A		Fragment B (modified form of A)	
S_1	y = (x - 1) * (x + 1)	S_1'	y = (x -1) * (x + 1)
S_2	if (y = 0)	S_2'	if (y = 0)
S_3	return (error)	S_3'	return (error)
S_4	else	S_4'	else
S_5	return $\left(\frac{1}{y}\right)$	S_5'	return $\left(\frac{1}{y-3}\right)$

Fig. 8: code fragment A and B

Software Maintenance

Test cases		
Test number	Input	Execution History
t_1	x = 1	S_1, S_2, S_3
t_2	x = -1	S_1, S_2, S_3
t_3	x = 2	S_1, S_2, S_5
t_4	x = 0	S_1, S_2, S_5

Fig. 9: Test cases for code fragment A of Fig. 8

Software Maintenance

If we execute all test cases, we will detect this divide by zero fault. But we have to minimize the test suite. From the fig. 9, it is clear that test cases t_3 and t_4 have the same execution history i.e. S_1, S_2, S_5. If few test cases have the same execution history; minimization methods select only one test case. Hence, either t_3 or t_4 will be selected. If we select t_4 then fine otherwise fault not found.

Minimization methods can omit some test cases that might expose fault in the modified software and so, they are not safe.

A safe regression test selection technique is one that, under certain assumptions, selects every test case from the original test suite that can expose faults in the modified program.

Software Maintenance

- **Selective Retest Techniques**

Selective retest techniques may be more economical than the "retest-all" technique.

Selective retest techniques are broadly classified in three categories :

1. **Coverage techniques : They are based on test coverage criteria. They locate coverable program components that have been modified, and select test cases that exercise these components.**

2. **Minimization techniques: They work like coverage techniques, except that they select minimal sets of test cases.**

3. **Safe techniques: They do not focus on coverage criteria; instead they select every test case that cause a modified program to produce different output than its original version.**

Software Maintenance

Rothermal identified categories in which regression test selection techniques can be compared and evaluated. These categories are:

Inclusiveness measures the extent to which a technique chooses test cases that will cause the modified program to produce different output than the original program, and thereby expose faults caused by modifications.

Precision measures the ability of a technique to avoid choosing test cases that will not cause the modified program to produce different output than the original program.

Efficiency measures the computational cost, and thus, practically, of a technique.

Generality measures the ability of a technique to handle realistic and diverse language constructs, arbitrarily complex modifications, and realistic testing applications.

Reverse Engineering

Reverse engineering is the process followed in order to find difficult, unknown and hidden information about a software system.

Software Maintenance

- **Scope and Tasks**

The areas there reverse engineering is applicable include (but not limited to):

1. **Program comprehension**
2. **Redocumentation and/ or document generation**
3. **Recovery of design approach and design details at any level of abstraction**
4. **Identifying reusable components**
5. **Identifying components that need restructuring**
6. **Recovering business rules, and**
7. **Understanding high level system description**

Software Maintenance

Reverse Engineering encompasses a wide array of tasks related to understanding and modifying software system. This array of tasks can be broken into a number of classes.

- **Mapping between application and program domains**

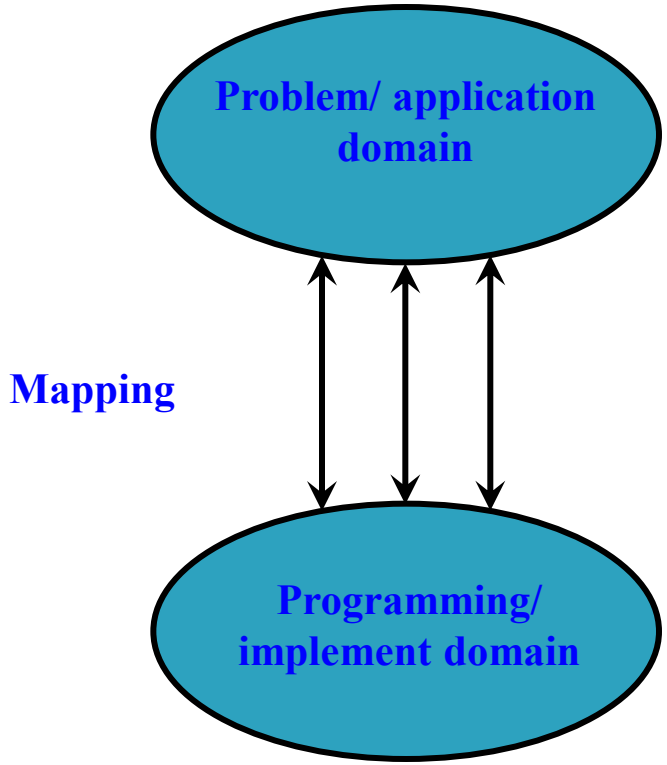


Fig. 10: Mapping between application and domains program

Software Maintenance

- **Mapping between concrete and abstract levels**
- **Rediscovering high level structures**
- **Finding missing links between program syntax and semantics**
- **To extract reusable component**

Software Maintenance

- **Levels of Reverse Engineering**

Reverse Engineers detect low level implementation constructs and replace them with their high level counterparts.

The process eventually results in an incremental formation of an overall architecture of the program.

Software Maintenance

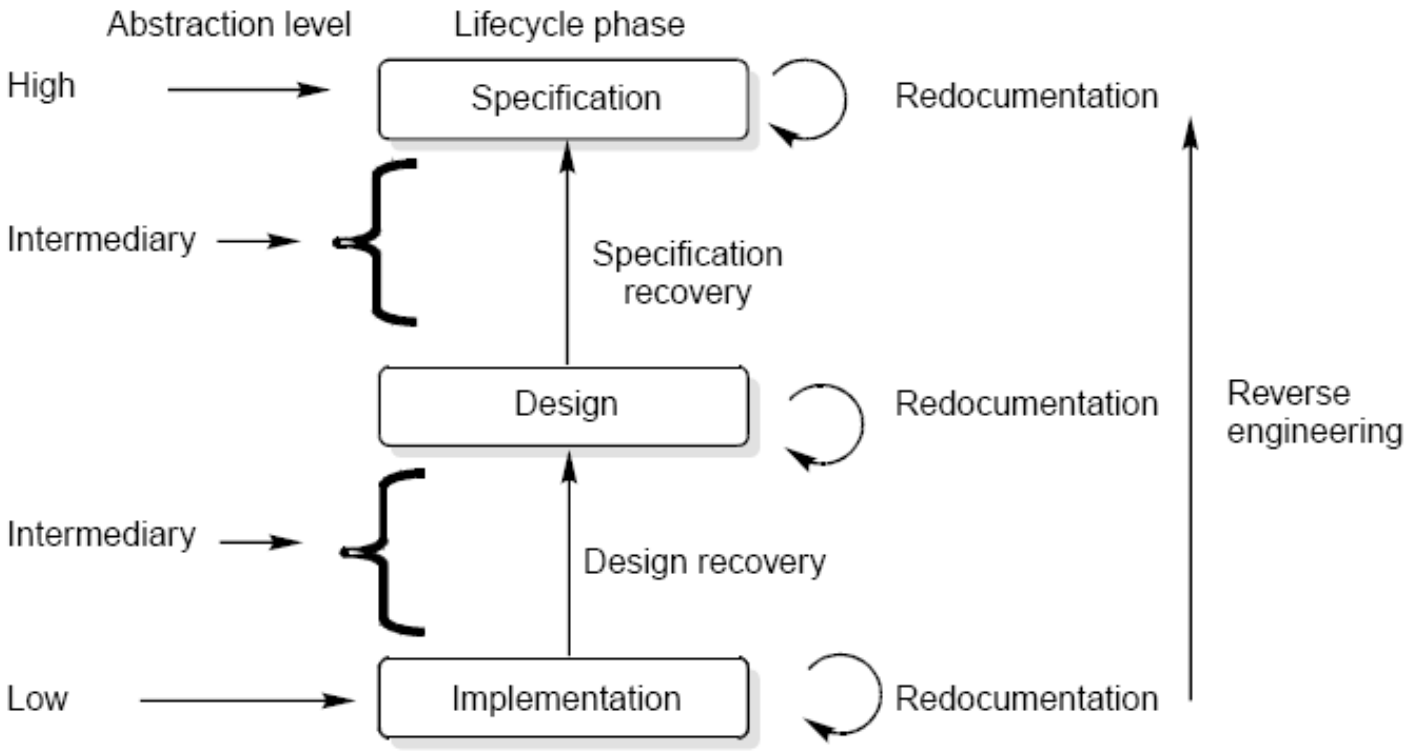


Fig. 11: Levels of abstraction

Software Maintenance

Redocumentation

Redocumentation is the recreation of a semantically equivalent representation within the same relative abstraction level.

Design recovery

Design recovery entails identifying and extracting meaningful higher level abstractions beyond those obtained directly from examination of the source code. This may be achieved from a combination of code, existing design documentation, personal experience, and knowledge of the problem and application domains.

Software Maintenance

Software RE-Engineering

Software re-engineering is concerned with taking existing legacy systems and re-implementing them to make them more maintainable.

The critical distinction between re-engineering and new software development is the starting point for the development as shown in Fig.12.

Software Maintenance

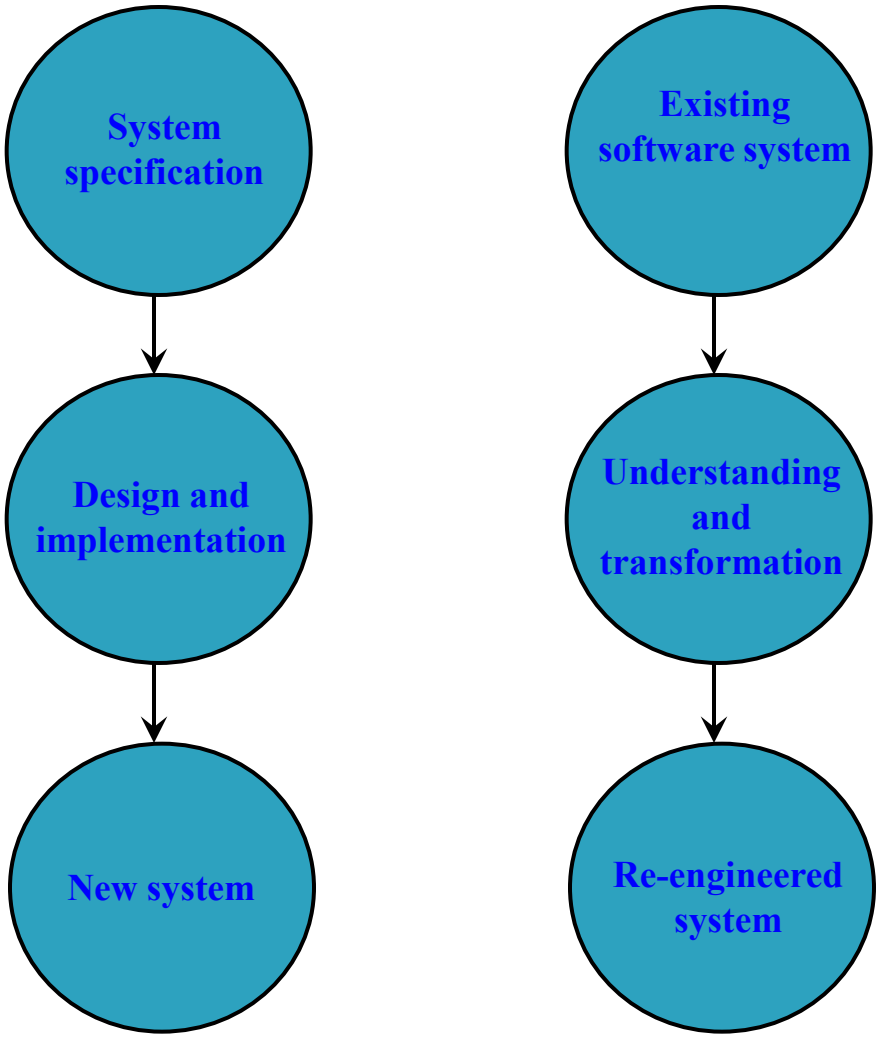


Fig. 12: Comparison of new software development with re-engineering

Software Maintenance

The following suggestions may be useful for the modification of the legacy code:

- ✓ **Study code well before attempting changes**
- ✓ **Concentrate on overall control flow and not coding**
- ✓ **Heavily comment internal code**
- ✓ **Create Cross References**
- ✓ **Build Symbol tables**
- ✓ **Use own variables, constants and declarations to localize the effect**
- ✓ **Keep detailed maintenance document**
- ✓ **Use modern design techniques**

Software Maintenance

- **Source Code Translation**

1. **Hardware platform update: The organization may wish to change its standard hardware platform. Compilers for the original language may not be available on the new platform.**

2. **Staff Skill Shortages: There may be lack of trained maintenance staff for the original language. This is a particular problem where programs were written in some non standard language that has now gone out of general use.**

3. **Organizational policy changes: An organization may decide to standardize on a particular language to minimize its support software costs. Maintaining many versions of old compilers can be very expensive.**

Software Maintenance

- **Program Restructuring**

1. **Control flow driven restructuring: This involves the imposition of a clear control structure within the source code and can be either inter modular or intra modular in nature.**

2. **Efficiency driven restructuring: This involves restructuring a function or algorithm to make it more efficient. A simple example is the replacement of an IF-THEN-ELSE-IF-ELSE construct with a CASE construct.**

Software Maintenance

```
IF Score > = 75 THEN Grade: = 'A'
ELSE IF Score > = 60 THEN Grade: = 'B'
ELSE IF Score > = 50 THEN Grade: = 'C'
ELSE IF Score > = 40 THEN Grade: = 'D'
ELSE IF Grade = 'F'
END
```

(*a*)

```
CASE Score of
75, 100: Grade: = 'A'
60, 74: Grade: = 'B';
50, 59: Grade: = 'C';
40, 49: Grade: = 'D';
ELSE Grade: = 'F'
END
```

(*b*)

Fig. 13: Restructuring a program

Software Maintenance

3. **Adaption driven restructuring: This involves changing the coding style in order to adapt the program to a new programming language or new operating environment, for instance changing an imperative program in PASCAL into a functional program in LISP.**

Software Maintenance

Configuration Management

The process of software development and maintenance is controlled is called configuration management. The configuration management is different in development and maintenance phases of life cycle due to different environments.

- **Configuration Management Activities**

The activities are divided into four broad categories.

1. **The identification of the components and changes**
2. **The control of the way by which the changes are made**
3. **Auditing the changes**
4. **Status accounting recording and documenting all the activities that have take place**

Software Maintenance

The following documents are required for these activities

- ✓ **Project plan**
- ✓ **Software requirements specification document**
- ✓ **Software design description document**
- ✓ **Source code listing**
- ✓ **Test plans / procedures / test cases**
- ✓ **User manuals**

Software Maintenance

- **Software Versions**

Two types of versions namely revisions (replace) and variations (variety).

Version Control :

A version control tool is the first stage towards being able to manage multiple versions. Once it is in place, a detailed record of every version of the software must be kept. This comprises the

- ✓ **Name of each source code component, including the variations and revisions**
- ✓ **The versions of the various compilers and linkers used**
- ✓ **The name of the software staff who constructed the component**
- ✓ **The date and the time at which it was constructed**

Software Maintenance

- **Change Control Process**

Change control process comes into effect when the software and associated documentation are delivered to configuration management change request form (as shown in fig. 14), which should record the recommendations regarding the change.

Software Maintenance

CHANGE REQUEST FORM

Project ID:

Change Requester with date:

Requested change with date:

Change analyzer:

Components affected:

Associated components:

Estimated change costs:

Change priority:

Change assessment:

Change implementation:

Date submitted to CCA:

Date of CCA decision:

CCA decision:

Change implementer:

Date submitted to QA:

Date of implementation:

Date submitted to CM:

QA decision:

Fig. 14: Change request form

Software Maintenance

Documentation

Software documentation is the written record of the facts about a software system recorded with the intent to convey purpose, content and clarity.

Software Maintenance

- **User Documentation**

S.No.	Document	Function
1.	System Overview	Provides general description of system's functions.
2.	Installation Guide	Describes how to set up the system, customize it to local hardware needs and configure it to particular hardware and other software systems.
3.	Beginner's Guide	Provides simple explanations of how to start using the system.
4.	Reference Guide	Provides in depth description of each system facility and how it can be used.
5.	Enhancement	Booklet Contains a summary of new features.
6.	Quick reference card	Serves as a factual lookup.
7.	System administration	Provides information on services such as net-working, security and upgrading.

Table 5: User Documentation

Software Maintenance

- **System Documentation**

It refers to those documentation containing all facets of system, including analysis, specification, design, implementation, testing, security, error diagnosis and recovery.

Software Maintenance

- **System Documentation**

S.No.	Document	Function
1.	System Rationale	Describes the objectives of the entire system.
2.	SRS	Provides information on exact requirements of system as agreed between user and developers.
3.	Specification/ Design	Provides description of: (i) How system requirements are implemented. (ii) How the system is decomposed into a set of interacting program units. (iii) The function of each program unit.
4.	Implementation	Provides description of: (i) How the detailed system design is expressed in some formal programming language. (ii) Program actions in the form of intra program comments.

Software Maintenance

S.No.	Document	Function
5.	System Test Plan	Provides description of how program units are tested individually and how the whole system is tested after integration.
6.	Acceptance Test Plan	Describes the tests that the system must pass before users accept it.
7.	Data Dictionaries	Contains description of all terms that relate to the software system in question.

Table 6: System Documentation