Software engineering
Assignment 4

Ques 1 discuss structure testing how is it different from functional testing

Definition of Structural Testing

Structural testing is a type of software testing that tests the internal code structure of the software. It verifies what happens inside the system and is also referred to as white-box or clear-box testing.

This type of testing requires good knowledge of the programming language and the code's internal implementation. It is based on how the system functions rather than the business requirements.

Different Types of Structural Testing Techniques

1. **Mutation** – A small change in the source code is made to test if the test cases can detect the changes. It is also called fault-based testing, as the fault is intentionally created in the code.
2. **Control flo**w – This type of structural testing uses the program's control flow as a model. Complete knowledge of the structure, design, and code of the software is necessary for this type of testing.
3. **Data flo**w – Data flow testing depends on data values and uses control flow graphs to identify any bugs that can interrupt data flow.

Functional testing is software testing that checks the functionality of the software application as per the requirement specification. It validates the output against the given input.

Functional testing aims to check the necessary usable function, input validation, and functional completeness. It is concerned with user experience and application function rather than the coding aspects.

Structural Testing	**Functional Testing**
This test evaluates the code structure or internal implementation of the code.	This test checks whether the software is functioning in accordance with functional requirements and specifications.
It is also known as white-box or clear-box testing as thorough knowledge and access of the code is required.	It is also known as black-box testing as no knowledge of the internal code is required.
Finds errors in the internal code logic and data structure usage.	It ensures that the system is error-free.
It does not ensure that the user requirements are met.	It is a quality assurance testing process ensuring the business requirements are met.
Performed the entire software in accordance with the system requirements.	Functional testing checks that the output is given as per expected.
Testing teams usually require developers to	A QA professional can simply perform this

perform structural testing.	testing.
Perform on low-level modules/software components.	The functional testing tool works on event analysis methodology.
It provides information on improving the internal structure of the application.	It provides information that prevents business loss.
Structural testing tools follow data analysis methodology.	Functional testing tool works on event analysis methodology.
Writing a structural test case requires understanding the coding aspects of the application.	Before writing a functional test case, a tester is required to understand the application's requirements.
It examines how well modules communicate with one another.	It examines how well a system satisfies the business needs or the SRS.

Ques2: differntitae bw integration testing and system testing

System Testing	Integration Testing
In system testing, we check the system as a whole.	In integration testing, we check the interfacing between the inter-connected components.
It is performed after integration testing.	It is performed after unit testing.
It is carried out for performing both functional and non-functional testing(performance, usability, etc).	It is generally limited to functional aspects of the integrated components.
Since the testing is limited to the evaluation of functional requirements, hence, it includes black-box testing techniques only.	Since the interfacing logic is required to perform this testing, hence, it requires white/grey box testing techniques along with black-box techniques.
The different types of system testing are- Functional testing, Performance testing, Usability testing, Reliability testing, Security testing, Scalability testing, Installation testing, etc.	The different approaches of performing integration testing namely – Top-down, bottom-up, big-bang and hybrid integration.

Ques 3 what are the various debugging approaches

Induction approach

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

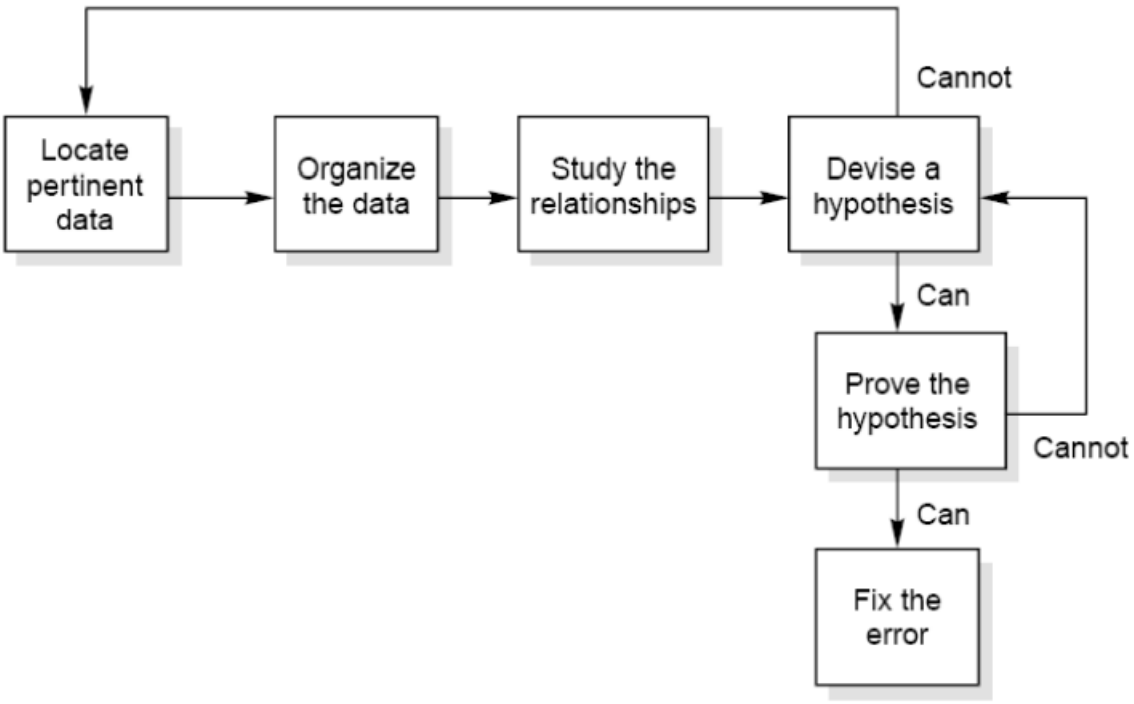


Fig. 32 : The inductive debugging process

Deduction approach

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

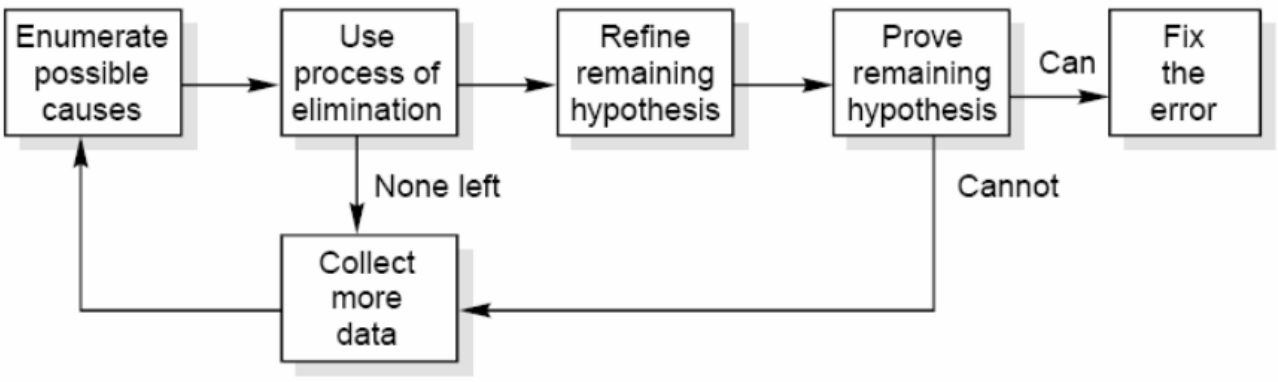


Fig. 33 : The inductive debugging process

Ques 4 : describe various maintenance cost estimation models

Belady and Lehman Model

The Belady and Lehman Model, proposed by Mehdi Belady and Mario M. Lehman, focuses on understanding software evolution and its impact on maintenance costs. This model emphasizes the relationship between software changes and various factors influencing these changes.

- Belady and Lehman Model

$$M = P + Ke^{(c-d)}$$

where

M : Total effort expended

P : Productive effort that involves analysis, design, coding, testing and evaluation.

K : An empirically determined constant.

c : Complexity measure due to lack of good design and documentation.

d : Degree to which maintenance team is familiar with the software.

Boehm Model

The Boehm Model, developed by Barry Boehm, focuses on software cost estimation, particularly maintenance costs. One of its key components is the **Annual Change Traffic (ACT)** metric, which helps estimate the effort required for maintenance based on the frequency and nature of changes. Important elements include:

- Boehm Model

Boehm used a quantity called Annual Change Traffic (ACT).

"The fraction of a software product's source instructions which undergo change during a year either through addition, deletion or modification".

$$ACT = \frac{KLOC_{added} + KLOC_{deleted}}{KLOC_{total}}$$

$$AME = ACT \times SDE$$

Where, **SDE :** Software development effort in person months

ACT : Annual change Traffic

EAF : Effort Adjustment Factor

$$AME = ACT * SDE * EAF$$

Ques 5 : what is regression testing differentiate bw regression and development testing

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

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.

Ques6 : Discuss what is reverse engineering and reengineering discuss ?

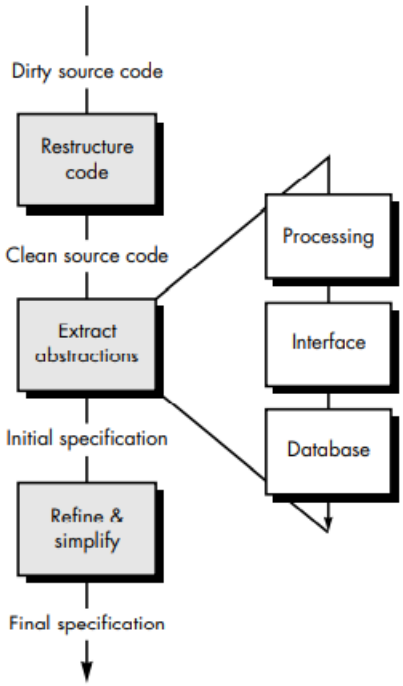
Reverse Engineering:

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

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

Reverse engineering can extract design information from source code, but the abstraction level, the completeness of the documentation, the degree to which tools and a human analyst work together, and the directionality of the process are highly variable [CAS88]. The abstraction level of a reverse engineering process and the tools used to effect it refers to the sophistication of the design information that can be extracted from source code. Ideally, the abstraction level should be as high as possible. That is, the reverse engineering

The reverse engineering process is represented in Figure 30.3. Before reverse engineering activities can commence, unstructured ("dirty") source code is restructured (Section 30.4.1) so that it contains only the structured programming constructs.2 This makes the source code easier to read and provides the basis for all the subsequent reverse engineering activities. The core of reverse engineering is an activity called extract abstractions. The engineer must evaluate the old program and from the (often undocumented) source code, extract a meaningful specification of the processing that is performed, the user interface that is applied, and the program data structures or database that is used.



- 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

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.

- Mapping between application and program domains

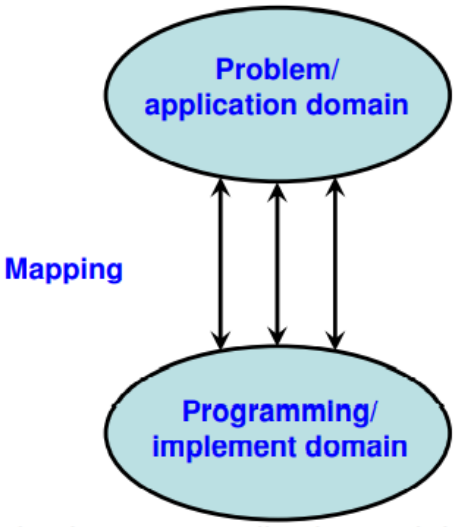


Fig. 10: Mapping between application and domains program

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 RE-Engineering

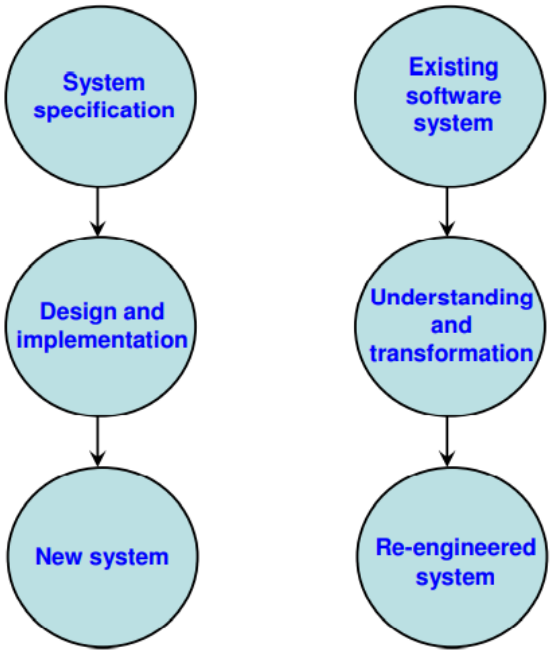


Fig. 12: Comparison of new software development with re-engineering

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 Figure 12.

Best Practices for Software Maintenance

To effectively maintain and re-engineer legacy systems, the following suggestions may be useful:

1. **Study Code Well Before Attempting Changes**:
    - Thoroughly understanding the existing codebase helps avoid introducing new bugs during modifications.
2. **Concentrate on Overall Control Flow and Not Coding**:
    - Focus on the overall structure and logic of the software to ensure that changes align with the intended functionality.
3. **Heavily Comment Internal Code**:
    - Detailed comments enhance code readability and provide context, facilitating easier understanding for future maintainers.
4. **Create Cross References**:
    - Cross-referencing related code segments helps developers understand dependencies and interactions within the system.

5. **Build Symbol Tables**:
    - Symbol tables assist in tracking variable definitions and scopes, making it easier to manage complex variable usage.
6. **Use Own Variables, Constants, and Declarations to Localize the Effect**:
    - Defining well-scoped variables reduces the risk of unintended side effects from changes.
7. **Keep Detailed Maintenance Documents**:
    - Comprehensive documentation of maintenance activities provides historical context and aids future maintenance efforts.