University of Sargodha

BS 4th Term Examination 2016

Subject: Computer Sc. & I.T **Paper: Software Engineering (CMP: 3310)**

Time Allowed: 2:30 Hours **Maximum Marks: 80**

Note: **Objective part is compulsory. Attempt any three questions from subjective part.**

Objective Part (Compulsory)

Q.1. Write short answers of the following in 2-3 lines each. (2*16)

i. What is SWOT Analysis?
ii. What is Software Refactoring?
iii. What is embedded system?
iv. What's the difference between Cohesion and Coupling?
v. Define SQA?
vi. What is Gantt Chart?
vii. Write down the name of different phases of Rational Unified Process?
viii. What's the difference between Testing and Debugging?
ix. Differentiate between Iterative Model and Incremental Model?
x. What is Extreme Programming?
xi. What is CMM?
xii. What is System Testing?
xiii. Differentiate between Project, Process and Product?
xiv. Define Stakeholder?
xv. What is Closed System?
xvi. Define SCM?

Subjective Part (3*16)

Q.2. Compare and Contrast between Incremental Process Model and Evolutionary Process Model?

Q.3. Define Software Design? Explain all sort of design in detail?

Q.4. A New Branch School District operates a fleet of 40 buses that serve approximately 1,000 students in grades 1 to 12. The bus operation involves 30 regular routes, plus special routes for activities, athletic events, and summer sessions. The district employs 12 fulltime drivers and 25 to 30 part-time drivers. A dispatcher coordinates the staffing and routes and relays messages to drivers regarding students and parents who call about pickup and drop-off arrangements.

a) Identify possible actors and use cases involved in school bus operations.
b) Create a use case diagram using any one of the use cases identified in part a
c) Create a sequence diagram for the use case you selected.

Q.5. Write a note on different types of software Testing techniques?

Q.6. What is Fact-Finding Technique? Explain different Fact-Finding Techniques with merits and demerits?

University of Sargodha

BS 4th Term Examination 2016

Subject: Comp Sc. & I.T **Paper: Software Engineering (CMP: 3310)**

****

Q1. Write short answers of the following.

i. What is SWOT Analysis?

SWOT analysis (strengths, weaknesses, opportunities and threats analysis) is a framework for identifying and analyzing the internal and external factors that can have an impact on the viability of a project, product, place or person.

SWOT analysis is most commonly used by business entities, but it is also used by nonprofit organizations and, to a lesser degree, individuals for personal assessment. Additionally, it can be used to assess initiatives, products or projects.

ii. What is Software Refactoring?

Refactoring is the process of altering an application's source code without changing its external behavior. The purpose of code refactoring is to improve some of the nonfunctional properties of the code, such as readability, complexity, maintainability and extensibility.

Refactoring can extend the life of source code, preventing it from becoming legacy code. The refactoring process makes future enhancements to such code a more pleasant experience. Refactoring is also known as **reengineering**.

iii. What is Embedded Systems?

An embedded system is a dedicated computer system designed for one or two specific functions. This system is embedded as a part of a complete device system that includes hardware, such as electrical and mechanical components.

Because an embedded system is engineered to perform certain tasks only, design engineers may optimize size, cost, power consumption, reliability and performance.

iv. What's the difference between Cohesion and Coupling?

Cohesion	Coupling
Cohesion is the indication of the relationship within module.	Coupling is the indication of the relationships between modules.
Cohesion shows the module's relative functional strength.	Coupling shows the relative independence among the modules.
Cohesion is a degree (quality) to which a component / module focuses on the single thing.	Coupling is a degree to which a component / module is connected to the other modules.
While designing you should strive for high cohesion i.e. a cohesive component/ module focus on a single task (i.e., single-mindedness) with little interaction with other modules of the system.	While designing you should strive for low coupling i.e. dependency between modules should be less.

v. Define SQA?

Software quality assurance (SQA) is a process that ensures that developed software meets and complies with defined or standardized quality specifications. SQA is an ongoing process within the software development life cycle (SDLC) that routinely checks the developed software to ensure it meets desired quality measures.

Solved by Talha Shahab

vi. What is Gantt Chart?

A Gantt chart is a horizontal bar chart developed as a production control tool in 1917 by Henry L. Gantt, an American engineer and social scientist. Frequently used in project management, a Gantt chart provides a graphical illustration of a schedule that helps to plan, coordinate, and track specific tasks in a project.

vii. Write down the name of different phases of Rational Unified Process?

The four phases of Rational Unified Process are as follows:

1) Inception
2) Elaboration
3) Construction
4) Transition

viii. What's the difference between Testing and Debugging?

Testing	Debugging
The purpose of testing is to find bugs and errors.	The purpose of debugging is to correct those bugs found during testing.
Testing is done by tester.	Debugging is done by programmer or developer.
It can be automated.	It can't be automated.
It can be done by outsider like client.	It must be done only by insider i.e. programmer.
Most of the testing can be done without design knowledge.	Debugging can't be done without proper design knowledge.

ix. Differentiate between Iterative and Incremental Model?

The Incremental approach is a method of software development where the model is designed, implemented and tested incrementally (a little more is added each time) until the product is finished. It involves both development and maintenance. The product is defined as finished when it satisfies all of its requirements

The Iterative Design is a design methodology based on a cyclic process of prototyping, testing, analyzing, and refining a product or process. Based on the results of testing the most recent iteration of a design, changes and refinements are made. This process is intended to ultimately improve the quality and functionality of a design.

x. What is Extreme Programming?

Extreme Programming (XP) is an agile software development framework that aims to produce higher quality software, and higher quality of life for the development team. XP is the most specific of the agile frameworks regarding appropriate engineering practices for software development. The general characteristics where XP is appropriate are:

- ✓ Dynamically changing software requirements
- ✓ Risks caused by fixed time projects using new technology
- ✓ Small, co-located extended development team
- ✓ The technology you are using allows for automated unit and functional tests

xi. What is CMM?

Capability Maturity Model (CMM) is a technical and cross-discipline methodology used to facilitate and refine software development processes and system improvement. Based on the Process Maturity Framework (PMF), CMM was developed to assess the performance capabilities of government contractors. Its levels are as follows:

1) Initial
2) Repeatable/Managed
3) Defined
4) Quantitatively Managed
5) Optimizing

xii. What is System Testing?

System Testing is the testing of a complete and fully integrated software product. Usually software is only one element of a larger computer based system. Ultimately, software is interfaced with other software/hardware

systems. System Testing is actually a series of different tests whose sole purpose is to exercise the full computer based system. System test falls under the black box testing category of software testing.

xiii. Differentiate between Process, Project and Product?

A software process specifies a method of development software. A software project, on the other hand is a development project in which a software process is used. And software products are the outcomes of a software project.

Each software development project starts with some needs and (hopefully) ends with some software that satisfies those needs. A software process specifies the abstract set of activities that should be performed to go from user needs to final product. The actual act of executing the activities for some specific user needs is a software project. And all the outputs that are produced while the activities are being executed are the products.

xiv. Define Stakeholder?

A stakeholder is a person or group who has an interest -- vested or otherwise -- in an enterprise and whose support is required in order for an enterprise to be successful. A stakeholder is a party that has an interest in a company and can either affect or be affected by the business. The primary stakeholders in a typical corporation are its investors, employees, customers and suppliers.

xv. What is Closed Systems?

A closed system is one, which doesn't interact with its environment. Such systems, in business world, are rare. Thus the systems that are relatively isolated from the environment but not completely closed are termed closed systems.

xvi. Define SCM?

Software configuration management (SCM) is a software engineering discipline consisting of standard processes and techniques often used by organizations to manage the changes introduced to its software products. SCM helps in identifying individual elements and configurations, tracking changes, and version selection, control, and baselining.

SCM is also known as software control management. SCM aims to control changes introduced to large complex software systems through reliable version selection and version control.

*Subjective Part (3*16)*

Q2. Compare and contrast between incremental Process Model and Evolutionary Process Model?

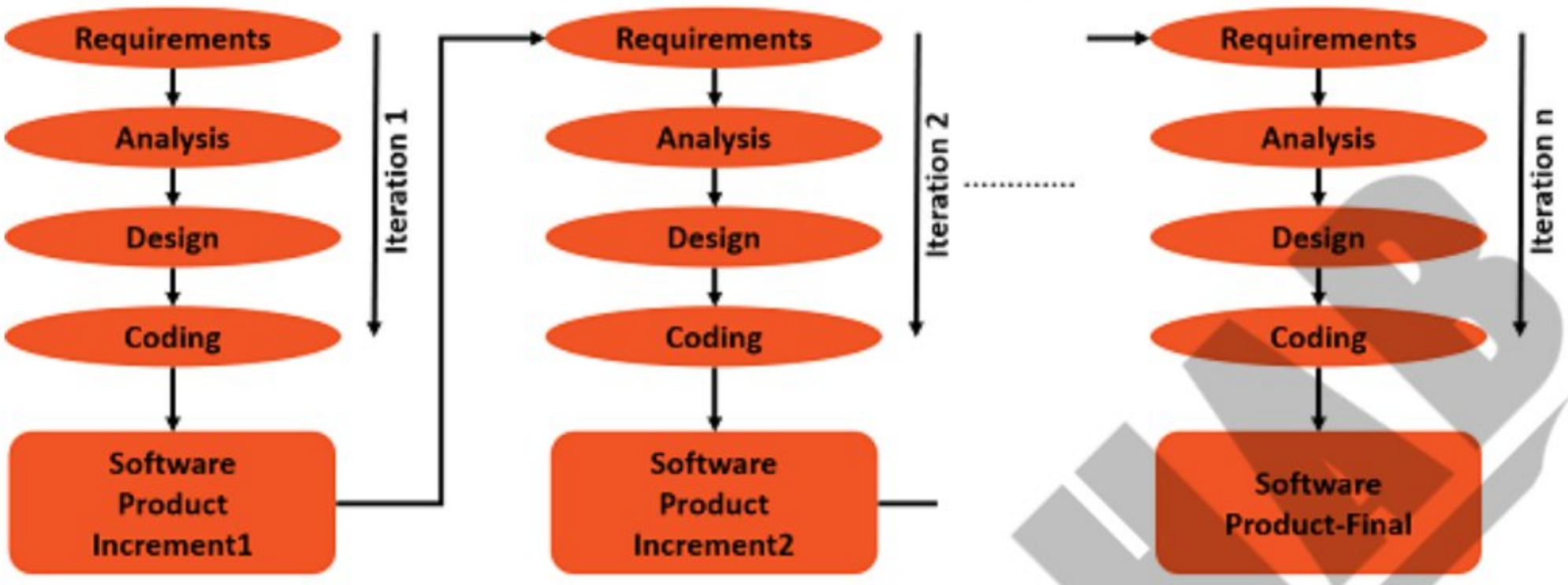
- **Incremental Process Model**

The incremental model delivers a series of releases, called increments that provide progressively more functionality for the customer as each increment is delivered. The incremental model combines elements of linear and parallel process flows *(at page 13)*.

"Incremental Model is a process of software development where requirements are broken down into multiple standalone modules of software development cycle. Incremental development is done in steps from analysis design, implementation, testing/verification, maintenance."

Solved by Talha Shahab

When an incremental model is used, the first increment is often a core product. That is, basic requirements are addressed but many supplementary features (some known, others unknown) remain undelivered. The core product is used by the customer (or undergoes detailed evaluation). As a result of use and/or evaluation, a plan is developed for the next increment. The plan addresses the modification of the core product to better meet the needs of the customer and the delivery of additional features and functionality. This process is repeated following the delivery of each increment, until the complete product is produced.



➢ Advantages of Incremental Process Model

The advantages or strengths of Iterative Incremental model are:

- ✓ You can develop prioritized requirements first.
- ✓ Initial product delivery is faster.
- ✓ Customers gets important functionality early.
- ✓ Lowers initial delivery cost.
- ✓ Each release is a product increment, so that the customer will have a working product at hand all the time.
- ✓ Customer can provide feedback to each product increment, thus avoiding surprises at the end of development.
- ✓ Requirements changes can be easily accommodated.

➢ Disadvantages of Incremental Process Model

The disadvantages of the Iterative Incremental model are:

- ✓ Requires effective planning of iterations.
- ✓ Requires efficient design to ensure inclusion of the required functionality and provision for changes later.
- ✓ Requires early definition of a complete and fully functional system to allow the definition of increments.
- ✓ Well-defined module interfaces are required, as some are developed long before others are developed.
- ✓ Total cost of the complete system is not lower.

When to Use Incremental Process Model?

Iterative Incremental model can be used when:

- ✓ Most of the requirements are known up-front but are expected to evolve over time.
- ✓ The requirements are prioritized.
- ✓ There is a need to get the basic functionality delivered fast.
- ✓ A project has lengthy development schedules.

Solved by Talha Shahab

- ✓ A project has new technology.
- ✓ The domain is new to the team.

➢ Evolutionary Process Models

Evolutionary process models produce an increasingly more complete version of the software with each iteration. Evolutionary models are iterative type models. They allow to develop more complete versions of the software.

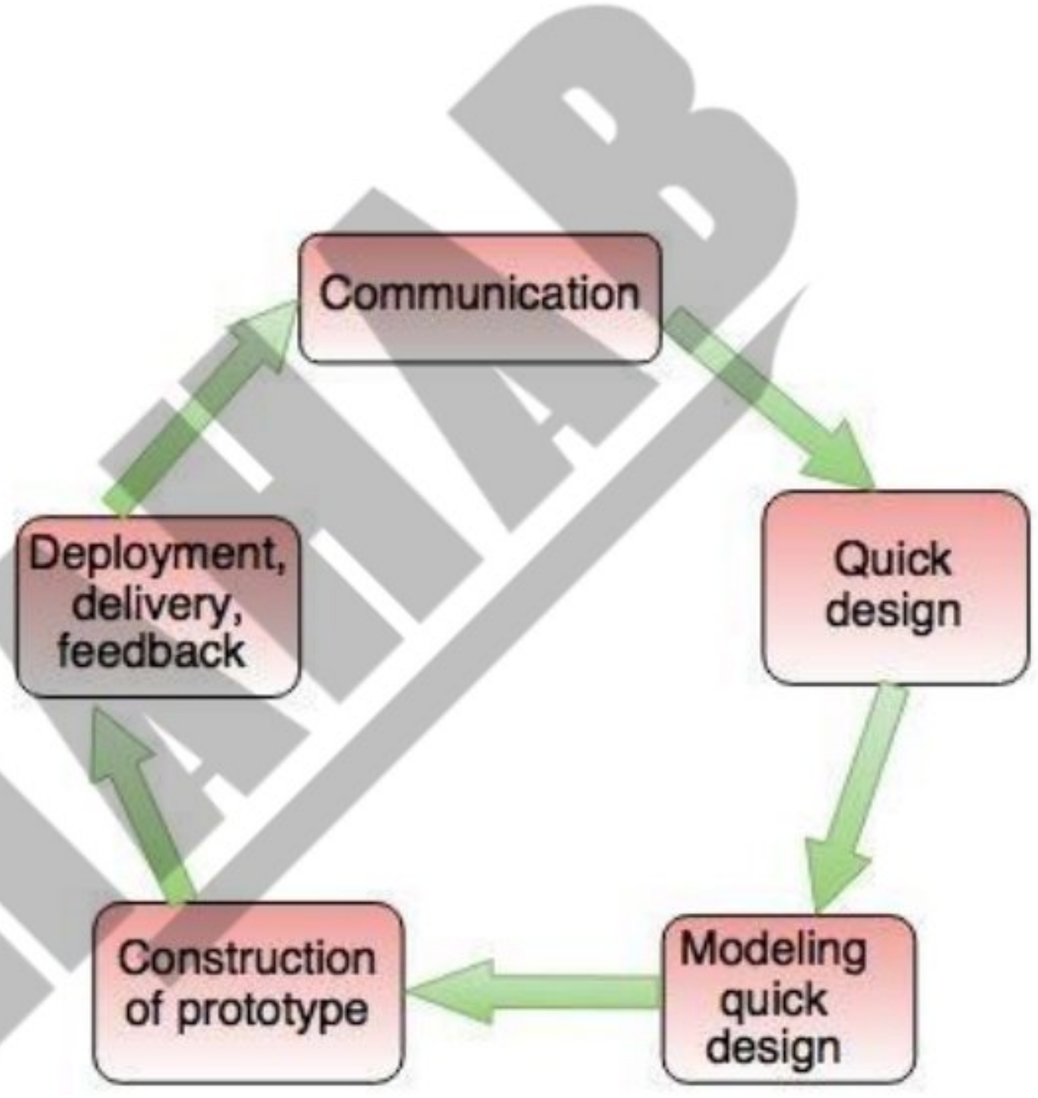
Following are the evolutionary process models.

A. **The prototyping model**
B. **The spiral model**
C. **Concurrent development model**

Their details are as follows:

A. The Prototyping model

Prototype is defined as first or preliminary form using which other forms are copied or derived. Prototype model is a set of general objectives for software. It does not identify the requirements like detailed input, output. It is software working model of limited functionality. In this model, working programs are quickly produced.



The different phases of Prototyping model are:

1. **Communication**: In this phase, developer and customer meet and discuss the overall objectives of the software.
2. **Quick design:** Quick design is implemented when requirements are known. It includes only the important aspects like input and output format of the software. It focuses on those aspects which are visible to the user rather than the detailed plan. It helps to construct a prototype.
3. **Modeling quick design:** This phase gives the clear idea about the development of software because the software is now built. It allows the developer to better understand the exact requirements.
4. **Construction of prototype**: The prototype is evaluated by the customer itself.
5. **Deployment, delivery, feedback:** If the user is not satisfied with current prototype then it refines according to the requirements of the user. The process of refining the prototype is repeated until all the requirements of users are met. When the users are satisfied with the developed prototype then the system is developed on the basis of final prototype.

Advantages of Prototyping Model

- ✓ Prototype model need not know the detailed input, output, processes, adaptability of operating system and full machine interaction.
- ✓ In the development process of this model users are actively involved.
- ✓ The development process is the best platform to understand the system by the user.
- ✓ Errors are detected much earlier.
- ✓ Gives quick user feedback for better solutions.
- ✓ It identifies the missing functionality easily. It also identifies the confusing or difficult functions.

Disadvantages of Prototyping Model:

- ✓ The client involvement is more and it is not always considered by the developer.
- ✓ It is a slow process because it takes more time for development.
- ✓ Many changes can disturb the rhythm of the development team.

Solved by Talha Shahab

✓ It is a thrown away prototype when the users are confused with it.

Q3. Define Software Design? Explain all sort of design in detail?

*{Topic no. 6.4 and 6.5 (User Interface Design and WebApps Interface Design) on page no. **49, 50** in notes}*

*{Page no. **317** and **335** in book "Software Engineering: A Practitioner's Approach 7th Edition by Roger S. Pressman}*

Q4. A New Branch School District Operates a fleet of 40 buses that serves approximately 1,000 students in grade 1 to 12. The bus operation involves 30 regular routes, plus special routes for activities, athletic events and summer sessions. The district employs 12 fulltime drivers and 25 to 30 part-time drivers. A dispatcher coordinates the staffing and routes and relays messages to drivers regarding students and parents who call about pickup and drop-off arrangements.

a. **Identify possible actors and use cases involved in school bus operation.**
b. **Create a use case diagram using any one of these use cases identified in part a.**
c. **Create a sequence diagram for the use case you selected.**

Answer:

a. Identify possible actors and use cases involved in school bus operation.

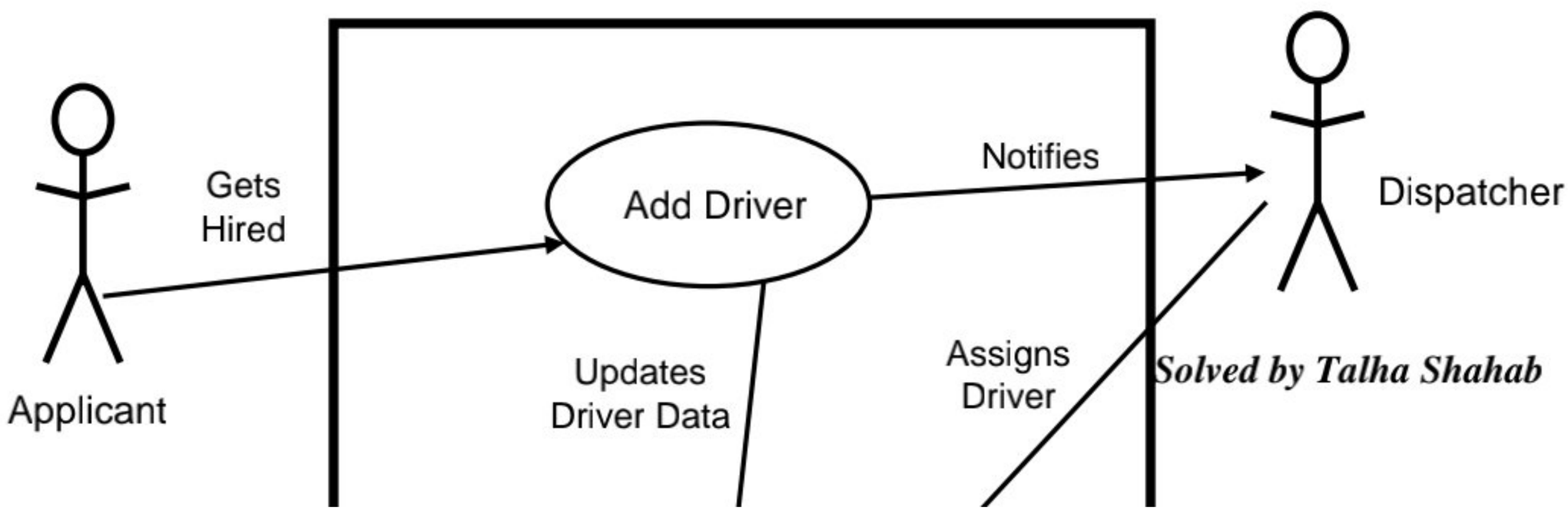
The actors in the above situation might include Applicant, Dispatcher, Driver, and Student. Use cases might include Add Driver, Add Route, Assign Student, and Assign Seat.

A sample use case diagram for the Add Driver use case follows:

Name:	Add Driver
Actor:	Dispatcher/Driver
Description:	This use case describes the process used to add a new bus driver assignment
Successful Completion:	Dispatcher adds driver to records Dispatcher assigns driver to available route Dispatcher notifies school, and driver of assignment Dispatcher gives keys to driver
Alternative:	Dispatcher adds driver to records Dispatcher assigns driver to substitute list Dispatcher notifies school, and driver of substitute assignment Dispatcher puts driver on waiting list for permanent route assignment
Pre-condition:	Applicant possesses appropriate license
Post-Condition:	Driver entered in records and given assignment
Assumptions:	None

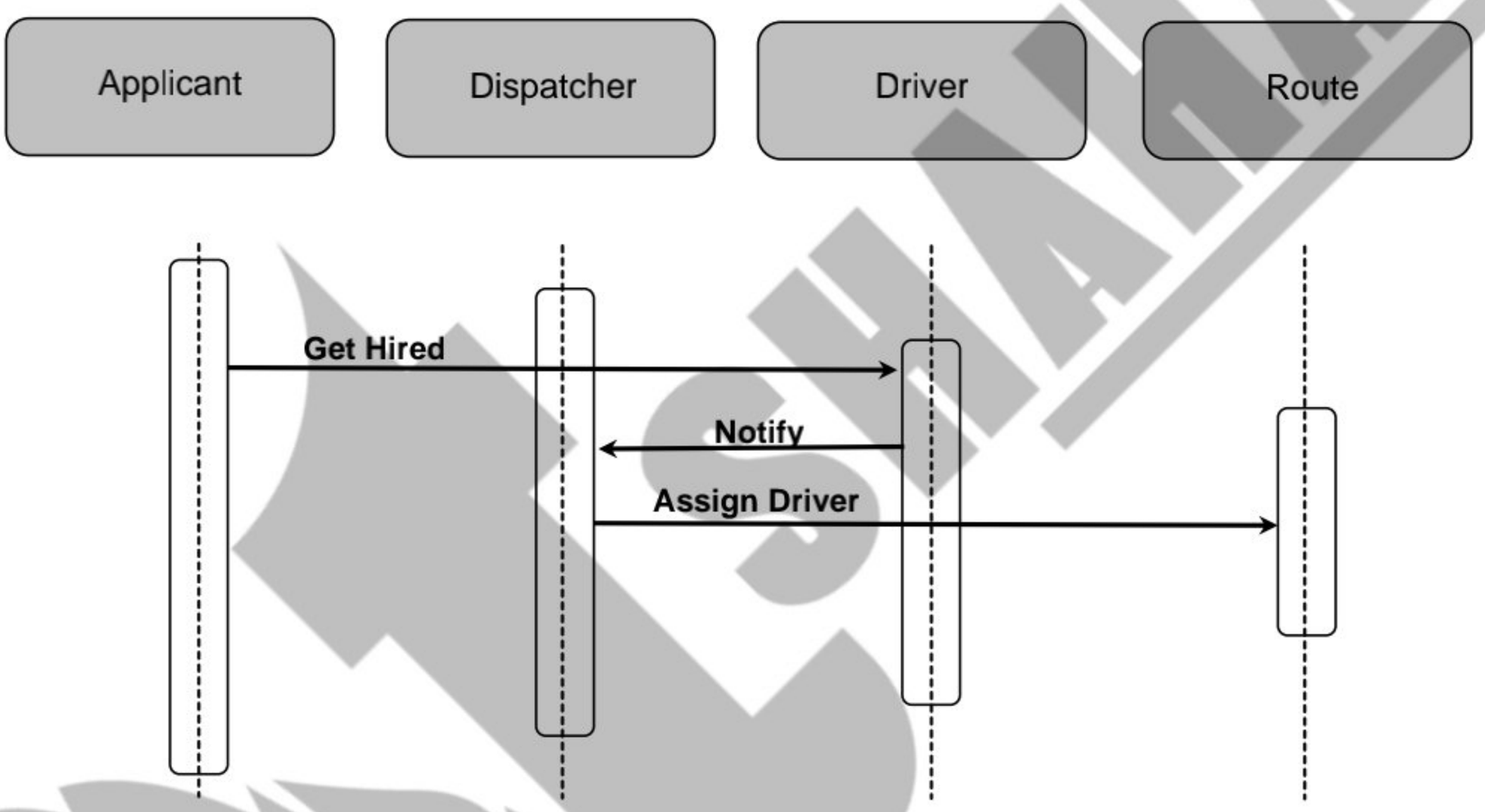
b. Create a use case diagram using any one of these use cases identified in part a.

A sample use case diagram for the Add Driver and Assign Route use cases follows:



c. Create a sequence diagram for the use case you selected.

A sample sequence diagram for the Add Driver use case follows:



Q5. Write a note on different types of software testing techniques?

Test Strategies for Conventional Software

- ☑ Unit Testing
- ☑ Integration Testing
- ☑ Regression Testing
- ☑ Smoke Testing

There are many strategies that can be used to test software. At one extreme, you can wait until the system is fully constructed and then conduct tests on the overall system in hopes of finding errors. This approach, although appealing, simply does not work. It will result in buggy software that disappoints all stakeholders. At the other extreme, you could conduct tests on a daily basis, whenever any part of the system is constructed. This approach, although less appealing to many, can be very effective. Unfortunately, some software developers hesitate to use it.

A testing strategy that is chosen by most software teams falls between the two extremes. It takes an incremental view of testing, beginning with the testing of individual program units, moving to tests designed to facilitate the

integration of the units, and culminating with tests that exercise the constructed system. Each of these classes of tests is described in the sections that follow.

Unit Testing

Unit testing focuses verification effort on the smallest unit of software design—the software component or module. Using the component-level design description as a guide, important control paths are tested to uncover errors within the boundary of the module. The relative complexity of tests and the errors those tests uncover is limited by the constrained scope established for unit testing. The unit test focuses on the internal processing logic and data structures within the boundaries of a component. This type of testing can be conducted in parallel for multiple components.

➢ Unit-test considerations.

Data flow across a component interface is tested before any other testing is initiated. If data do not enter and exit properly, all other tests are moot. In addition, local data structures should be exercised and the local impact on global data should be ascertained (if possible) during unit testing.

Selective testing of execution paths is an essential task during the unit test. Test cases should be designed to uncover errors due to erroneous computations, incorrect comparisons, or improper control flow.

Boundary testing is one of the most important unit testing tasks. Software often fails at its boundaries. That is, errors often occur when the nth element of an n-dimensional array is processed, when the *ith* repetition of a loop with *i* passes is invoked, when the maximum or minimum allowable value is encountered. Test cases that exercise data structure, control flow, and data values just below, at, and just above maxima and minima are very likely to uncover errors.

Among the potential errors that should be tested when error handling is evaluated are:

(1) Error description is unintelligible,
(2) Error noted does not correspond to error encountered,
(3) Error condition causes system intervention prior to error handling,
(4) Exception-condition processing is incorrect,
(5) Error description does not provide enough information to assist in the location of the cause of the error.

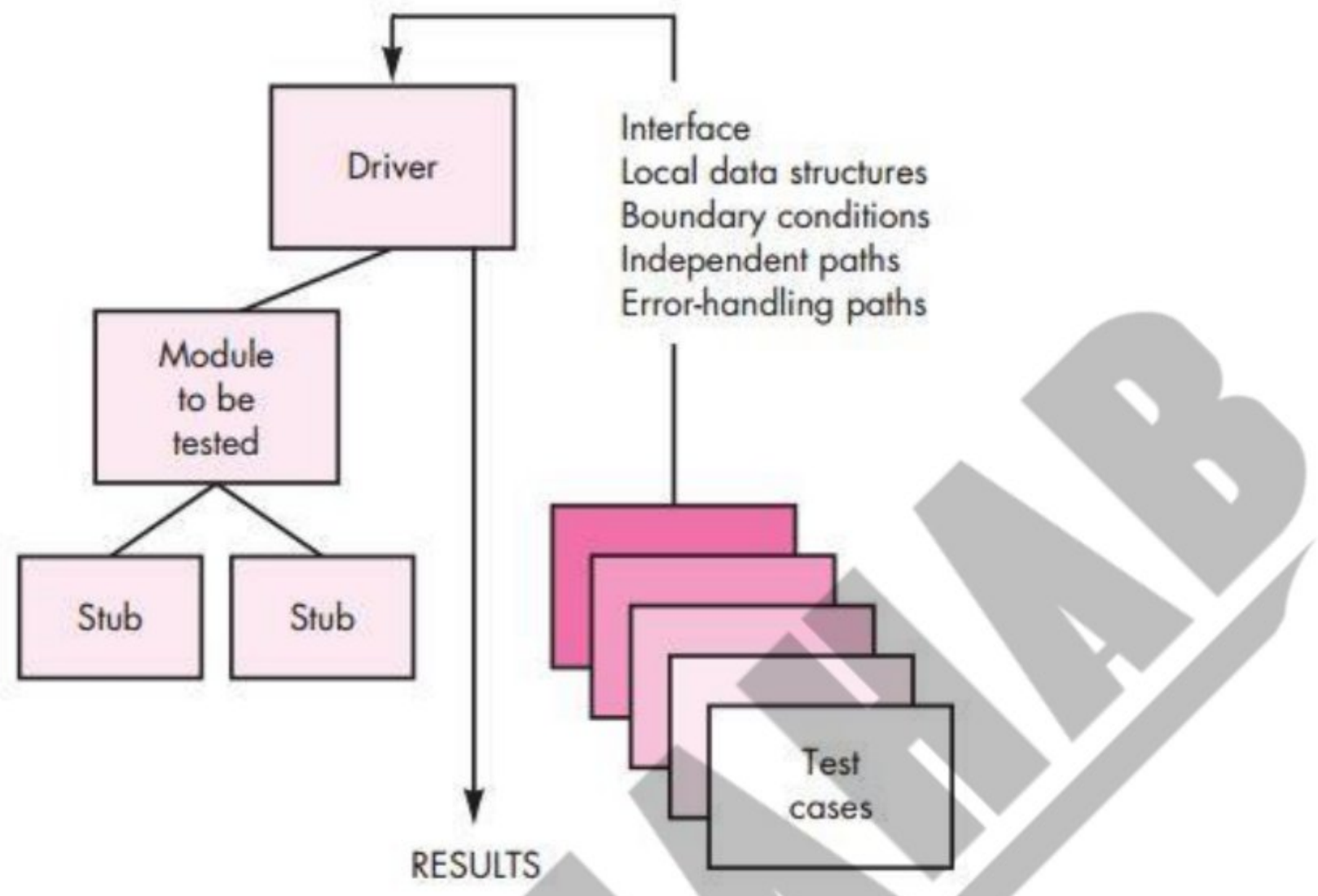
➢ Unit-test procedures.

Unit testing is normally considered as an adjunct to the coding step. The design of unit tests can occur before coding begins or after source code has been generated. A review of design information provides guidance for establishing test cases that are likely to uncover errors in each of the categories discussed earlier. Each test case should be coupled with a set of expected results.

Because a component is not a stand-alone program, driver and/or stub software must often be developed for each unit test. In most applications a driver is nothing more than a “main program” that accepts test case data, passes such data to the component (to be tested), and prints relevant results. Stubs serve to replace modules that are subordinate (invoked by) the component 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 control to the module undergoing testing.

Drivers and stubs represent testing “overhead.” That is, both are software that must be written (formal design is not commonly applied) but that is not delivered with the final software product. If drivers and stubs are kept simple, actual overhead is relatively low. Unfortunately, many components cannot be adequately unit tested with “simple” overhead software. In such cases, complete testing can be postponed until the integration test step (where drivers or stubs are also used).

Unit testing is simplified when a component with high cohesion is designed. When only one function is addressed by a component, the number of test cases is reduced and errors can be more easily predicted and uncovered.



Integration Testing

Integration testing is a systematic technique for constructing the software architecture while at the same time conducting tests to uncover errors associated with interfacing. The objective is to take unit-tested components and build a program structure that has been dictated by design.

There is often a tendency to attempt non-incremental integration; that is, to construct the program using a "big bang" approach. All components are combined in advance. The entire program is tested as a whole. And chaos usually results! A set of errors is encountered. Correction is difficult because isolation of causes is complicated by the vast expanse of the entire program. Once these errors are corrected, new ones appear and the process continues in a seemingly endless loop.

Incremental integration is the antithesis of the big bang approach. The program is constructed and tested in small increments, where errors are easier to isolate and correct; interfaces are more likely to be tested completely; and a systematic test approach may be applied. In the paragraphs that follow, a number of different incremental integration strategies are discussed.

➢ Top-down integration

Top-down integration testing is an incremental approach to construction of the software architecture. Modules are integrated by moving downward through the control hierarchy, beginning with the main control module (main program). Modules subordinate (and ultimately subordinate) to the main control module are incorporated into the structure in either a depth-first or breadth-first manner. The integration process is performed in a series of five steps:

(1) The main control module is used as a test driver and stubs are substituted for all components directly subordinate to the main control module.
(2) Depending on the integration approach selected (i.e., depth or breadth first), subordinate stubs are replaced one at a time with actual components.
(3) Tests are conducted as each component is integrated.
(4) On completion of each set of tests, another stub is replaced with the real component.

Solved by Talha Shahab

(5) Regression testing (discussed later in this section) may be conducted to ensure that new errors have not been introduced.

The process continues from step 2 until the entire program structure is built. The top-down integration strategy verifies major control or decision points early in the test process. In a "well-factored" program structure, decision making occurs at upper levels in the hierarchy and is therefore encountered first. If major control problems do exist, early recognition is essential. If depth-first integration is selected, a complete function of the software may be implemented and demonstrated. Early demonstration of functional capability is a confidence builder for all stakeholders.

Top-down strategy sounds relatively uncomplicated, but in practice, logistical problems can arise. The most common of these problems occurs when processing at low levels in the hierarchy is required to adequately test upper levels. Stubs replace low-level modules at the beginning of top-down testing; therefore, no significant data can flow upward in the program structure. As a tester, you are left with three choices:

(1) Delay many tests until stubs are replaced with actual modules,
(2) Develop stubs that perform limited functions that simulate the actual module,
(3) Integrate the software from the bottom of the hierarchy upward.

The first approach (delay tests until stubs are replaced by actual modules) can cause you to lose some control over correspondence between specific tests and incorporation of specific modules. The second approach is workable but can lead to significant overhead, as stubs become more and more complex. The third approach, called bottom-up integration is discussed in the paragraphs that follow.

➢ Bottom-up integration

Bottom-up integration testing, as its name implies, begins construction and testing with atomic modules (i.e., components at the lowest levels in the program structure). Because components are integrated from the bottom up, the functionality provided by components subordinate to a given level is always available and the need for stubs is eliminated. A bottom-up integration strategy may be implemented with the following steps:

(1) Low-level components are combined into clusters (sometimes called builds) that perform a specific software sub-function.
(2) A driver (a control program for testing) is written to coordinate test case input and output.
(3) The cluster is tested.
(4) Drivers are removed and clusters are combined moving upward in the program structure.

Regression testing

Each time a new module is added as part of integration testing, the software changes. New data flow paths are established, new I/O may occur, and new control logic is invoked. These changes may cause problems with functions that previously worked flawlessly. In the context of an integration test strategy, regression testing is the re-execution of some subset of tests that have already been conducted to ensure that changes have not propagated unintended side effects.

In a broader context, successful tests (of any kind) result in the discovery of errors, and errors must be corrected. Whenever software is corrected, some aspect of the software configuration (the program, its documentation, or the data that support it) is changed. Regression testing helps to ensure that changes (due to testing or for other reasons) do not introduce unintended behavior or additional errors.

- ✓ A representative sample of tests that will exercise all software functions.
- ✓ Additional tests that focus on software functions that are likely to be affected by the change.
- ✓ Tests that focus on the software components that have been changed.

Solved by Talha Shahab

As integration testing proceeds, the number of regression tests can grow quite large. Therefore, the regression test suite should be designed to include only those tests that address one or more classes of errors in each of the major program functions. It is impractical and inefficient to re-execute every test for every program function once a change has occurred.

Smoke testing

Smoke testing is an integration testing approach that is commonly used when product software is developed. It is designed as a pacing mechanism for time-critical projects, allowing the software team to assess the project on a frequent basis. In essence, the smoke-testing approach encompasses the following activities:

(1) Software components that have been translated into code are integrated into a build. A build includes all data files, libraries, reusable modules, and engineered components that are required to implement one or more product functions.

(2) A series of tests is designed to expose errors that will keep the build from properly performing its function. The intent should be to uncover "showstopper" errors that have the highest likelihood of throwing the software project behind schedule.

(3) The build is integrated with other builds, and the entire product (in its current form) is smoke tested daily. The integration approach may be top down or bottom up.

The daily frequency of testing the entire product may surprise some readers. However, frequent tests give both managers and practitioners a realistic assessment of integration testing progress. Smoke testing provides a number of benefits when it is applied on complex, time critical software projects:

- ✓ Integration risk is minimized. Because smoke tests are conducted daily, incompatibilities and other show-stopper errors are uncovered early, thereby reducing the likelihood of serious schedule impact when errors are uncovered.
- ✓ The quality of the end product is improved. Because the approach is construction (integration) oriented, smoke testing is likely to uncover functional errors as well as architectural and component-level design errors. If these errors are corrected early, better product quality will result.
- ✓ Error diagnosis and correction are simplified. Like all integration testing approaches, errors uncovered during smoke testing are likely to be associated with "new software increments"—that is, the software that has just been added to the build(s) is a probable cause of a newly discovered error.
- ✓ Progress is easier to assess. With each passing day, more of the software has been integrated and more has been demonstrated to work. This improves team morale and gives managers a good indication that progress is being made.

Q6. What is Fact-Finding technique? Explain different Fact-Finding techniques with merits and demerits?

To study any system the analyst needs to do collect facts and all relevant information. The facts when expressed in quantitative form are termed as data. The success of any project is depended upon the accuracy of available data. Accurate information can be collected with help of certain methods/ techniques. These specific methods for finding information of the system are termed as fact finding techniques. Interview, Questionnaire, Record View and Observations are the different fact finding techniques used by the analyst. The analyst may use more than one technique for investigation.

Interview

This method is used to collect the information from groups or individuals. Analyst selects the people who are related with the system for the interview. In this method the analyst sits face to face with the people and records their responses. The interviewer must plan in advance the type of questions he/ she is going to ask and should be ready to answer any type of question. He should also choose a suitable place and time which will be

comfortable for the respondent. The information collected is quite accurate and reliable as the interviewer can clear and cross check the doubts there itself. This method also helps gap the areas of misunderstandings and help to discuss about the future problems. Structured and unstructured are the two sub categories of Interview. Structured interview is more formal interview where fixed questions are asked and specific information is collected whereas unstructured interview is more or less like a casual conversation where in-depth areas topics are covered and other information apart from the topic may also be obtained.

Questionnaire

It is the technique used to extract information from number of people. This method can be adopted and used only by a skillful analyst. The Questionnaire consists of series of questions framed together in logical manner. The questions are simple, clear and to the point. This method is very useful for attaining information from people who are concerned with the usage of the system and who are living in different countries. The questionnaire can be mailed or send to people by post. This is the cheapest source of fact finding.

Record View

The information related to the system is published in the sources like newspapers, magazines, journals, documents etc. This record review helps the analyst to get valuable information about the system and the organization.

Observation

Unlike the other fact finding techniques, in this method the analyst himself visits the organization and observes and understand the flow of documents, working of the existing system, the users of the system etc. For this method to be adopted it takes an analyst to perform this job as he knows which points should be noticed and highlighted. In analyst may observe the unwanted things as well and simply cause delay in the development of the new system.