UNIT 5

DEVELOPING

PART-A

SHORT QUESTIONS WITH SOLUTIONS

Q1. What are the nine practices that helps in maintaining the code clean?

Answer : Model Paper-II, Q1(i)

The following are the nine practices that can make code easy to understand and easy to change,

1. **Incremental Requirements**

   It enables team to start the work when the customers work on requirements details.

2. **Customer Tests**

   It helps in conveying tricky domain rules.

3. **Test-driven Development**

   It enables programmers to ensure that their code will generate the output as desired.

4. **Refactoring**

   It allows the programmers to enhance the quality of the code without modifying its functionality.

5. **Simple Design**

   It supports modifications in the design to add new features irrespective of the level of modification.

6. **Incremental Design and Architecture**

   It enables the programmers to work on features and technical framework concurrently.

7. **Spike Solutions**

   It gives information by making use of controlled experiments.

8. **Performance Optimization**

   It drives the optimization efforts by means of hard data.

9. **Exploratory Testing**

   It allows tester to determine the gaps in the team's way of thinking about processes.

Q2. Define test driven development.

Answer : Model Paper-III, Q1(i)

A Test-driven Development (TDD) is a software development approach that undergoes a quick cycle of coding, testing and refactoring processes. While adding features, a pair performs these type of several cycles thereby implementing and verifying the software small steps till there is nothing to add or subtract. Most of the researches reveal that TDD mainly decreases the occurrences of faults (Janzen and Saiedian). It improves design, documents public interfaces and prevents from future mistakes. It is obvious that TDD is not perfect. But, it is difficult to employe TDD on legacy code bases, TDD can be used in almost any project.

Q3. What does refactoring mean?

Answer :

Refactoring can be viewed as a process of modifying the code design without changing the behaviour of the class. When it is implemented it stays the same. In some cases, one form is better than another form. Thus making it reversible. For example, the expression can be changed from $a^2 - b^2$ to $(a + b)(a - b)$ and vice versa. Similarly, the design of the code can be changed. Thus, the problem of entropy can be resolved.

Q4. What is risk driven architecture?

Answer : **Model Paper-I, Q1(i)**

A risk driven architecture is a model that directs the software developers to design architecture by applying minimum set of architecture techniques that are enough to reduce their most critical risk. It prevents the developers from spending their time on low- impact techniques. The key element of risk driven architecture is to ensure that the risk can be identified clearly and does not mislead the developers.

Q5. What is spike solution?

Answer : **Model Paper-III, Q1(j)**

A spike solution is a small technical experiment that is used to explore solutions for a specific problem. It can be used in a number of situations which are as follows,

1. It helps the team in getting familiarized with new hardware or software.
2. It helps to evaluate a problem thoroughly and helps the development team to divide work among themselves individually.
3. It helps in reducing the risk occurrence in future and discover the issues/bugs that were left unnoticed in the previous iteration.

Q6. Define charter.

Answer :

A charter is a one page simple document that is created at the initial phase of project development. It serves as a plan or blueprint to achieve the desired goal of the project. It consists of three quality attributes: stability, reliability and performance. Charters are similar to stories and they function well when written down.

Q7. Write about heuristics.

Answer : **Model Paper-II, Q1(j)**

Heuristic is a technique that supports in decision making. The testers need not have to spend time in assessing all the potential options, conducting cost benefit analysis and ranking the alternatives, rather they can employ heuristics to arrive at the best option. The testers can improved the exploratory efforts by making and maintaining a list of essential heuristics when software exploring is performed. This same list can be used as a welcome reference for the programmers because they executes the software.

Q8. What are the mistakes that developers generally commit while testing a software?

Answer : **Model Paper-I, Q1(j)**

Generally, making mistakes while testing a software or planning a testing effort is prevailing. But, one of the most common mistakes in customer tests is defining what happens in the user interface instead of providing examples of business rules. For instance, to illustrate an account rep should not delete a mailed invoice, one might commit the mistake of writing the following,

1. Logging in as an account rep
2. Creating new invoice
3. Entering data
4. Saving invoice
5. E-mail invoice to customer
6. Verifying whether the invoice should be deleted (or) not.

It is essential to focus on the bottom line of the rules instead of thinking how a specific rule would work in when the team implements this, it becomes more easier, compact, and faster to run.

PART-B

ESSAY QUESTIONS WITH SOLUTIONS

5.1 INCREMENTAL REQUIREMENTS

Q9. Describe briefly about software development. Explain the nine practices to keep the code clean.

Answer : Model Paper-II, Q10(a)

Team cooperation is the key component for development of a successful software. Each and every member of the project development team plays a specific role in overall management of software development process i.e., testers help the team by reducing the risk of bugs occurrence whereas programmers roll out the cost of technical infrastructure over the life cycle of the project. When the members of a team are not dedicated, competent and engaged in the effort then projects are not successful. The cost of developing the software can be reduced by improving the quality of design and coding. A well-formed project can repay its investment and a high quality of code reduces the cost of development in both short-scale projects and long-scale projects.

The following are the nine practices that can make code easy to understand and easy to change,

1. **Incremental Requirements**

   It enables team to start the work when the customers work on requirements details.

2. **Customer Tests**

   It helps in conveying tricky domain rules.

3. **Test-driven Development**

   It enables programmers to ensure that their code will generate the output as desired.

4. **Refactoring**

   It allows the programmers to enhance the quality of the code without modifying its functionality.

5. **Simple Design**

   It supports modifications in the design to add new features irrespective of the level of modification.

6. **Incremental Design and Architecture**

   It enables the programmers to work on features and technical framework concurrently.

7. **Spike Solutions**

   It gives information by making use of controlled experiments.

8. **Performance Optimization**

   It drives the optimization efforts by means of hard data.

9. **Exploratory Testing**

   It allows tester to determine the gaps in the team's way of thinking about processes.

Q10. Discuss briefly about the sources of incremental requirements.

Answer :

The following are the sources of incremental requirements.

1. **The Living Requirements Document**

In XP, initially the development team performs face-to-face interaction with online customers to know about their requirements for the project. The team acquires complete details about the requirements from the customers itself, instead of looking in a document. This method of communication is much more efficient than written communication. It consumes less time and also eliminates the need of long requirements analysis phase. In XP, expert customers are considered as the key to make requirements analysis successful. Real customers, experts play a vital role in making a decision. When customers involve other team members before making a decision then selection of options becomes easy. For instance, if a customer involve programmer in an user interface discussion then he/she can attain balance between splendid UI and reduced implementation cost.

Always, note down the requirements that you may forget. This help the customers in reminding themselves of the decisions they have made earlier and also makes answering the questions of the developers easy.

2. Work Incrementally

The project development team should work on the software by dividing the tasks individually. This helps the team in ensuring that the work performed by other team member does not get affected by any kind of issue in other team member's performance. This in turn makes easier and it can be done concurrently with the other team member's work.

3. Vision, Features and Stories

Initially, the development processing by analyzing the vision of the project, features and stories as defined in "Release Planning".

4. Rough Expectations

Member must be prepared with the answers for the queries that programmers might raise before they start developing the project. A rough sketch of their expectations can facilitate the programmers while implementation.

5. Mock-ups, Customer Tests and Completion Criteria

Resolve the details for every story before the programmers begins their implementation. Generate rough mock-ups indicating your expectations from the project, create customers tests that gives examples of tricky domain concepts and define the meaning of "done done" for every story.

6. Customer Review

Each and every story must be reviewed thoroughly while they are under development and before the stories are "done done". It does not need to exhaustively test the software. Although, customers can depend on the programmers to test their work, but customer should review the areas where the programmers may think in other way than the customer does such as screen layout, interactions and terminology between screen elements. This may help the customers in identifying the errors that have occurred due to misconception. If the errors detected are minor then customers can interact with the programmers about that changes that must be made. But if the identified errors are major then prepare story cards for these modifications. Therefore, in this way programmers will learn about the customer's expectations for the application over the period of time.

5.2 CUSTOMER TESTS

Q11. Describe the process to create customer tests.

Answer : Model Paper-II, Q11

Customer Test

Customer tests are the tests that are performed for communication between the customers and programmers. This practice helps the programmers to understand all the domain rules well enough such that the application meets all the customer requirements.

To generate customer tests, the following describe, demonstrate and develop processes are used.

1. Describe

In the beginning of this iteration, customers check their stories and determine whether there are any concepts which programmers may not understand.

For instance, if one of the stories of customer is "Allow involve deleting", the customer does not define how to delete invoices because the programmers understands the meaning of deleting something. So, customers provide examples that explains in which situation it is fine to delete an invoice. When the customer identifies potential misunderstandings they bring the team together at a whiteboard and summarize the story where they describe briefly how the story have to work.

However, the task of invoices deletion is not as easy as it looks because the programmers have to maintain an auditing trail. There can be number of rules behind this issues. One of the rules is that if an invoice has not been sent and gets deleted then it is not audited because this kind of invoice was a mistake. This kind of rules make difficult for the programmers to implement.

2. Demonstrate

Tables are considered as the most simplest from of source to describe any information. So, by using tables different scenarios are demonstrated.

Example

Consider the following invoice that has not been sent to customers, so an Account Rep can be deleted.

Sent	User Name	OK to Delete
No	Account Rep	Yes

Anyone can delete - CSRs, managers and admins.

Sent	User Name	To Delete
No	CSR	Yes
No	Manager	Yes
No	Admin	Yes

But, once if the above invoice is sent, then it can be deleted only by admins and managers and then it is audited.

Sent	User Name	To Delete
Yes	Account Rep	No
Yes	CSR	No
Yes	Manager	Audited
Yes	Admin	Audited

It is not a simple situation of whether something is sent or not because, "sent" specifies several conditions. If the invoice is seen by the customers, it is considered as "Sent". In this case, it can be deleted only by admins (or) managers.

Sent	User Name	To Delete
Printed	Accountant	No
Posted to Web	Accountant	No
Exported	Accountant	No
Emailed	Accountant	No

3. Develop

When the customers have completed certain amount of stories then, they start documenting them. So that programmers can start working on implementing their rules. During this process, the customers can also evaluate whether the format of the examples work well with respect to the automated testing or not. The customers must not formalize their examples at early stage until they have worked on all the examples. So, once the customers provides few examples, programmers must start implementing the code using normal test driven development. Although, the customer test can also be used for writing the test by the programmers they must employ TDD only as it provides more better fine grained support.

Q12. What is the role of a customer in customer testing?

Answer :

In few development teams, programmers and testers perform all the activities of customer testing without any involvement of the customers. In such teams, customers serve only as a mute observer. But, if customers do not participate in customer testing then it may cause difficulties for the programmers in development of a valuable software. The process of speaking and listening to the customers allows programmers to design and create a product that meets all the user requirements. If programmers want to achieve high user satisfaction then it is essential to gain understanding of how customers user their product.

Sometimes customer does not show interest in taking the lead. This problem can be overcomed if programers and testers ask customer's help saying that they need domain expertise. They can ask customers to join them in discussion and must be sure to take advantages of it. Besides customers, testers also play a vital role in customer testing. They facilitate by identifying bugs, detecting errors, or any other issue that end user may come across. Through customers lead the effort, testers benefit from their technical expertise and ability to suggest different ideas. Both developers and testers must maintain the standards specified by the customers in mind while creating or upgrading their products or services.

Q13. Write about the tool used for automating the customers examples.

Answer :

The programmers can use any tool for transforming customer's examples into automated tests. "Word Cunningham's Frame work for Integrated test (Fit)" is specifically designed for this purpose. It enables customer to use HTML for integrating tables and descriptions, and then it runs the tables using programmer created fixtures for executing the tests. Moreover, it also enables customers to review, execute and even extend their own tests. Although the programmers have to write the fixtures, customers can easily modify the existing table to test the idea using FIT tool. It also enables the testers to modify the tests as an assistance to exploratory testing. Since, the tests are written in HTML, HTML editor can be used to modify the tests. The programmers do not make fit very complex. It is used as a simple tool.

For instance, the following example of invoice auditing use a custom column fixture. The columns of the table represents a method (or) variable in the fixture.

Example Fixture of C#

```
public class InvoiceAuditingFixture : ColumnFixture
{
    public Status sent;
    public UserRole invoiceuser;
    public Permission ToDelete( )
    {
        InvoiceAuditer auditer = new InvoiceAuditer(User, StatusofInvoice) return auditer. DelPermission;
    }
}
```

Making use of FIT, requires a ubiquitous language and good design. In this case, whole value objects with dedicated domain layer works best. If customers encounter any problem using fit, they must approach mentor to discuss about modifications required in the design.

5.3 TEST DRIVEN DEVELOPMENT

Q14. What is the test driven development? Explain its need.

Answer : **Model Paper-III, Q10(a)**

Test-Driven Development

A Test-Driven Development (TDD) is a software development approach that undergoes a quick cycle of coding, testing and refactoring processes. While adding features, a pair performs these type of several cycles thereby implementing and verifying the software small steps till there is nothing to add or subtract. Most of the researches reveal that TDD mainly decreases the occurrences of faults (Janzen and Saiedian). It improves design, documents public interfaces and prevents from future mistakes. It is obvious that TDD is not perfect. But, it is difficult to employe TDD on legacy code bases, TDD can be used in almost any project.

Need of TDD

Earlier, when punch cards were used by the programmers they used to vigorously hand check their code so as to ensure that the code would compile. Even a single error while compiling would result in failed batch jobs and intense debugging so as to search for the misplaced character. But, at present compiling code is not a big task. The IDEs check the syntaxes at the same time when the user types code and also compiles it each time when saved. Bugs and errors existing in the code can be identified quickly and easily using feedback loops. It implements the same principle to programmer intent. Just like, modern compilers generate more feedback on the syntax of the code. It rapidly provides feedback on the code execution. It verifies whether the code performs accordingly or not for every 20 or 30 seconds. This makes detecting and fixing the bugs more easier. TDD enables to define the same idea in multiple ways i.e., firstly, with a test and secondly with production code. If they match, then it specifies both of them are coded correctly. If they didn't match then there is a mistake at some point.

In Test-Driven Development (TDD), the tests are defined from the perspective of a class public interface. Instead of focusing on implementation, programmers focuses on behavior of the class. i.e., they prefer on writing the test prior to the production code. This helps them on generating interfaces which are easy to use rather than to implement, which helps to improve interface design.

After the completion of TDD, the tests that remains are checked with the rest of the code and acts as a documentation of the code. Moreover, the programmers run all the tests with every construct to make sure that code works continuously as designed. If the code's behavior is changed accidentally then the tests fail and bug is signaled.

Q15. How Test Driven Development (TDD) is used?

Answer :

A driven development is employed by following the "red, green, refactor" cycle. It takes less than five minutes for completion of one cycle. This cycle is repeated until the task gets completed. The following steps are involved in a TDD cycle,

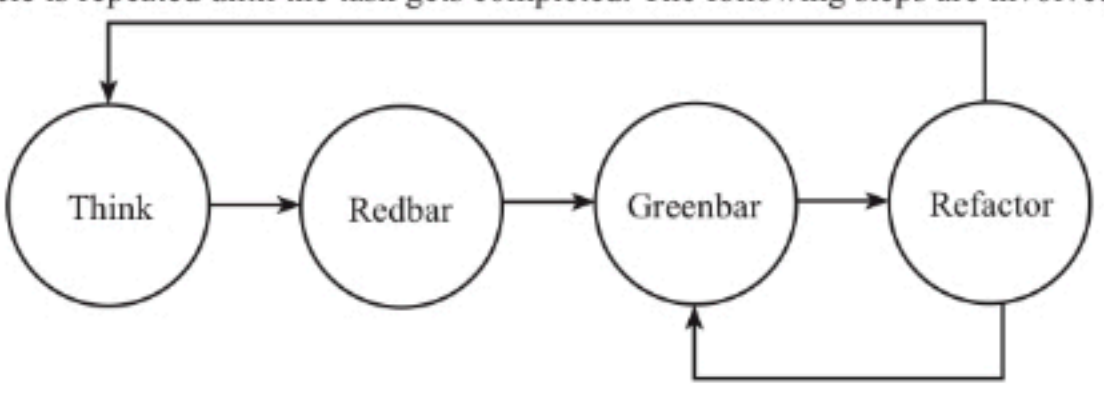


Figure: TDD (Test-Driven Development) Cycle

Step 1 : Think

In XP, there is a saying that "Don't write any production code unless you have a failing test". TDD employs small test to write the code which is enough to pass the tests. Initially, programmer must think about how the code should be, after that he must think about small increment that needs less than five lines of code.

Now, programmer should think about the test, which will also contain few lines of code of code only. This is the most difficult task of TDD because it is not easy to think is small increments driving your code backward. But, this task can be accomplish by pair programming which enables to think about the tests that leads the code to the next level increment .

Step 2 : Red Bar

In this step, write the test and sufficient code for current increment which less than five lines of code. It is okay to add more lines but next programmer time should try for a smaller increment. Coding should be done in terms of the class behavior and its public interface.

After coding the test, run the entire collection of tests and it displays the failed test. Mostly in TDD testing tool, result will be in red progress bar.

The programmers first opportunity is to differentiate their intent with reality. When the test fails in another way, or the test does not fail, it specifies that either the programmer's test is defective or it does not work according to the programmer's intent. The programmer should always detect the problems and should be able to project what the code is upto.

Step 3 : Green Bar

Now, the programmer must write sufficient production code to pass the test. Again, the programmer requires less than five lines of code. The programmer should do what is required to pass the test rather than focusing on the purity of design. Run the tests again and see all the passed tests, which results in a green progress bar.

The programmer's second opportunity is to differentiate their intent with reality. But, if the test fails, go back and try to develop the possible good code as early as possible. If the programmer can detect the problem after checking the code for second time then it must be resolved and if the problem cannot be identified then erase the new code and try again. Generally, it is better to delete the new test and again start the cycle with smaller increment.

Step 4 : Refactor

When all the tests are passed refactor them. Review the code and check for necessary improvements. For every problem, refactor the code to fix it. The work should be done in small refactorings not longer than few minutes and run the tests one after the other. The tests should always pass. If the test fails and the answer is not clear then perform the refactoring again and get back to known good code. Refactor possible number of times and improve the design to the best possible level.

Step 5 : Repeat

The cycle is started again when the programmer is prepared to add new features. With the completion of each TDD cycle a small part of well tested, well designed code is added. The small increment made in TDD are the key to success. The programmer will run number of cycles rapidly. The the programmers spend more time on refactoring one or more cycles and then it speeds up again.

The programmer can complete more than 20 cycles in 60 minutes with practice. The programmer should not focus on speed of cycle because it causes the programmer to skip design and refactoring. So, programmer must run the tests repeatedly, using small increments and reduce the time required with a red bar.

Q16. Describe test driven development and legacy code.

Answer :

Test Driven Development (TDD) and Legacy Code

The Legacy code is basically referred to as "code without tests". Actually, this code is not designed for the purpose of testing. To modify the code, introduce the tests. This makes legacy code more complex to use.

When the programmer do not want to modify the existing code then he/she may face the difficulty to eliminate the code debt. Therefore, it becomes more complex to understand the whole picture. Legacy code has frequent set of code debt and subsequently, these functions and methods might produce adverse effect which go unnoticed.

This problem can be solved by implementing end-to-end smoke tests. In doing so, these tests makes use of common scenarios which have the component that the programmer can easily modify. However, with this the programmer won't be able to make big difference but it can alert the programmer with respect to this. More importantly, unit test can be implemented inspite of smoke test. But, the problem which arises here is that, the programmer will not find any individual components to tests because the legacy code is usually tightly coupled code. Rather it is better to search for ways to test strategically interrupt the program execution.

Searching and utilizing more than its capability entails complex code. In this situation, the code is temporarily made erroneous, so that the programmer makes the code error-free. After performing the tests, perform refactoring of code to make it test-friendly. Improve the tests, and then ensure that they are not very complex, then do normal test driven development.

Q17. Write short notes on focused integration tests.

Answer :

When unit tests becomes insufficient, the code must be communicate (or) integrate with other code using TDD. When the code communicates across the network, it makes use of file system and it is said to be integration test. The integration test is the process which communicates with the database, uses the file system, communicate throughout the network by leaving the marks of its own process. Most ideal integration test is focused integration tests that interacts with the outside world. The main challenge that the programmer faces while using integration tests is the necessity to conduct test on external dependency. The programmer respective of order, must run the test in the same manner. Although this process is easy than unit tests but becomes complex with integration tests.

Ensure that every integration tests runs completely on its own. In doing so, environment should be set-up according to its requirements and after sometime restore to the previous environment. This pattern should be continued even if an exception is thrown or if the test fails. Subsequently, complexity arises if test fails. The programmer does not need to carryout multiple integration tests. The ideal integration test is the one have narrow focus, in essence it verifies only one characteristics of the program's capability to communicate the outside database. The programmes must ensure that various integration tests in a set of tests must be proportional to the different types of external interactions that holds. But, it does not consider the full size of the program. The test are same to the full size of the program. It multiple integration tests are needed then it indicates the design problems. It implies that the code which communicates the outside world is not connective. Therefore, the integration tests are needed by database class to communicate with database.

Q18. Explain in brief about end-to-end test.

Answer :

End-to-End Tests

The combination of focused integration tests and unit tests effectively runs in terms of code and tests. The programmer should be capable of such modifications in code and after modifying the code, the programmer should not just be able to detect any errors and also correct them.

Inorder to know whether unit test and integration test has assimilated property or not, the end-to-end tests consume large system space which initiates at user interface, goes through the business layer, uses the database and then return. The examples of end-to-end tests are functional tests and acceptance tests. It is also referred as integration tests.

Eventhough, end-to-end tests make the programmers assured that their coding is error free but they faces many problems.

Typically, these tests are difficult develop because they demand certain procedures which are error prone, labor intensive and break down.

In addition to this, each time the data is modified. Instead of running multiple tests every second, they consumes seconds (or) minutes for one test. They supply inaccurate security, because they use different branches in the coding, which makes it difficult to analyze like it is hard to speak about which sections of the code are really covered exploratory testing to verify the capability of unit tests and integration tests. Each time a problem is detected by exploratory testing, instead of introducing end-to-end tests, use the information to improve the approach to unit tests and integration tests.

The limitations of design might inhibit unit tests and integration tests from effective code testing. It occurs when the programmer have legacy code. Try to avoid end-to-end tests, use unit tests and integration tests.

Q19. Explain the following terms,

(i) Testing tools

(ii) Unit tests.

Answer :

(i) Testing Tools

A testing framework is required for using Test-Drivan Development (TDD). For this purpose, mostly used frameworks are XUnit Tools which is open source, NUnit (.NET) and JUnit(Java). If an XUnit tool is not installed in the system then the programmers can manually develop their own tool. With the existing tools, the user can make use of GUIs and other features. Although this is not compulsory, but is important to ensure that all test methods should run in the form of a single set. Some assert methods and an explicit outcome indicating the pass (or) fail at the time of test execution.

(ii) Unit Tests

The unit tests emphasizes on class (or) method. This testing tool be runs at high speed in the memory. They should be capable of running 100 unit tests per second, based upon the platform. Ideally, the unit can be viewed as,

1. Unit tests runs at high speed. But, if the tests does not runs fast, then typically they are not referred as unit tests.

2. The test is not said to be a unit test, if it qualifies the following,

   (a) If it communicates to the database.

   (b) If it communicates in the network.

   (c) If it uses file system.

   (d) If the programmer runs unit test and modifies the things to environment such as making editing configuration files.

   Therefore, the tests which adopts the above points is typically referred to as integration tests, but not unit tests.

   Inorder to develop a unit test, a good design is needed. If the programmer uses high coupled system then writing unit tests becomes difficult.

   If the programmer faces problem in doing this then, it indicates that the design has defects. Then find the ways to decouple the code so that each and every classes might be tested separately.

The Importance of Speed

The programmer by implementing TDD, get numerous distinct feature to discover the following the steps red, green, refactor the programmer can obtain results in a consistent manner.

Test speed is one of the district feature of Test-Driven Development (TDD). It is not concern with frequency of every single increment, but certainly emphasize on the time it consumes to run all test. Subsequently, the tests are run in TDD frequently in every minute i.e., one or more times. The tests should be fast. However, if the tests fail to run at high speed they cause lot of difficulties.

All the group of tests are sequentially run to ensure that nothing is broken. This should be done while the process of refactoring and before integrating that may affect multiple classes. The following are three basic types of automated tests,

1. The unit tests runs at a rate of hundreds/second.
2. The focused integration tests runs at a rate of handful per second.
3. End-to-End tests frequently needs seconds/test.

5.4 REFACTORING

Q20. Define refactoring. Discuss various terms involved in it.

Answer : **Model Paper-I, Q10(a)**

Refactoring

Refactoring can be viewed as a process of modifying the code design without changing the behaviour of the class. When it is implemented it stays the same. In some cases, one form is better than another form. Thus making it reversible. For example, the expression can be changed from $a^2 - b^2$ to $(a + b)(a - b)$ and vice versa. Similarly, the design of the code can be changed. Thus, the problem of entropy can be resolved.

Terms in Refactoring

The following are the terms of refactoring,

1. Reflective Design

Refactoring allows the programmer to design an approach called reflective design. In this, both designing and coding is possible. Subsequently, analysis of existing code design is also possible. Most efficient methods to determine the improvements is by using code smalls which assists in identifying the common problems in design.

On the other hand, the code smell does not mean that the design is problematic. It detects the problem present in it. When the programmer faces any problem, they try to solve them.

2. Divergent Change and Shotgun Surgery

These two situations are basically meant to determined the cohesion problems in the code.

"Divergent change" takes place when irrelevant changes occur in the same class. This shows that the class has many approaches and it is better to divide it, each approach should have its own space to function. "Shortgun surgery" is the reverse of divergent change, it takes place when the programmer have to modify the multiple classes to sustain modification in a single plan. This shows that the approach is present in multiple areas in the code. So, search and create a single property for the plan.

3. Primitive Obsession and Data Clumps

The developers can encompass concepts of high-level design along with the help of primitive type. For instance, a decimal number may represent errors. This signifies primitive obsession code smell. The problem can be resolved by using encapsulation.

"Data clumps" are similar to primitive obsession, they take place when one or many primitives represents a concept as a group.

For instance, many lines of code many represents an address. It is not encapsulated of being summarized in a single class rather the data is grouped together. Each time group of variables are passed around the programmer may face problem of data clumping. In a way similar to primitive obsession here also the approach is encapsulated in a single class.

4. Data Class and Wannabee Static Class

The most basic mistakes in object-oriented design occur when programmers stores their code and data in different classes. This results in the generation of multiple copies of code. When the programmer have a data class that is higher than instance variables, it is combined with getters and setters. Subsequently, if the class consists of methods without meaning per object state then it is said to be wannabee state class.

The major strengths of object - oriented programming is the ability to merge data with the code which operates on that data. Ideally, data classes and wannabee statics both are initially together and divided. So now rejoin them by merging instance variables with methods that works on those variables.

5. Coddling Nulls

A specific challenge to programmer given by null references are fruitful. Sometimes, they represent invalid states and other error conditions. Initially, the programmers are not sure about how to deal when they encounter null reference. In essence programmers methods consistently look for null references and returns null to themselves.

This type of coddling null results in a faulty software and complex methods. Here, the errors along with null cascade operate on the software counting uncertain failures while execution. Also sometimes the null gets access to the database which results into reoccurrence of application failures.

Use fail fast strategy rather than coddling nulls and consequently null should not be used as a parameter for any method, or constructor until it clearly defines the semantics. Instead of returning null throw exceptions to mark errors. Ensure that null reference that is unpredicted can cause the code to throw an exception. This is caused as a result of dereferenced or it is clearly declared.

6. Time Dependencies and Half-baked Objects

The time dependencies happens each time a class and methods invokes a particular order. Here, half-baked object must be constructed first then analyzed with a method call and then used. These objects are special type of time dependency.

Time dependencies consistently represents a problem of encapsulation. So instead of managing its own state by itself, the expectations of class is that some of its own states should be managed by the callers. This entails duplicating of code and bugs in callers. In a way similar to other method, encapsulate the class state sufficiently by finding more methods. Sometimes, the programmer observes that the class has number of responsibilities and can advantage by dividing into multiple classes.

Q21. Discuss in brief about process of analyzing existing code.

Answer :

Analyzing Existing Code

In reflective design, the programmer needs to understand the detailed information regarding the design of an existing code. This task can be accomplished connectively by dedicating the task to the team member. Additionally, a decision or meeting with respect to this is necessary.

In particular, if the team members fail to understand the design then the programmer may have to do work on the code manually. In doing so, the programming should focus on every major component of interactions and responsibilities. The tools for unit testing ASP.NET code-behind logic is NUnitASP. There are many classes, one of it is Httpclient, in which the programmer may make calls to web server of ASP.NET.

To ensure about the assumption, check the class names, methods and instance variables. In particular, the HttpClient contains methods such as FollowLink, GetPage, SubmitForm and HasCookie with some related methods and properties and some of the USER_AGENT constants. It is clear that the HttpClient uses a web browser.

Extend the understanding towards related elements and draw the understanding on board.

Design a UML sequence diagram that can assistance understanding the interaction of individual methods with the rest of the system. The programmer can initiate with a method as they want to understand and also check every line one by one. Subsequently every called method is added recursively to the sequence diagram. It is used when the programmer is confused about the working of method, otherwise it is very time consuming.

Ideally, programmer should not require these techniques regularly, because these techniques are used to understand the unknown code design. The programmer should have good knowledge about the parts of the software, should be capable enough to communicate with someone in the team who have good software knowledge. Until the programmers are working the team, team should not have any trouble in dealing with the design of existing code unless the programmer operates on legacy code.

But, if the programmer still needs these techniques regularly, it means something is incorrect.

Q22. Explain how to implement the process of refactoring.

Answer : **Model Paper-I, Q11**

Reflective design typically enables us to understand the necessary changes to be made. So, refactoring provides the ability to implement those changes. Each time the programmer refactor, it proceeds in a series of small conversions. Each conversion is also known as refactoring. Every refatoring makes use of Rubik's Cube.

It improves the quality of an application's code without affecting its functionality. Sometimes, the process of refactoring cannot be understood by members who are new to it. Because, the fact that refactoring not just involves redesigning the code, but also focuses on making that changes in a series of controlled steps gets lost on people. Every step should be executed in a short time period and tests must pass after each step. However, there are various forms of individual refactorings. But, the programmers need not remember all the individual refactorings. Rather they must just understand the concept behind refactoring. Every change in the design of code requires a series of refactorings. So, once the developer understands how to change their design through these sequence of refactorings, he can make a vivid change in his code thereby resulting in good and efficient coding.

The following program to develop an HTTP query string parser illustrates refactoring in Action,

```
public class QueryStrTest extends TestCase
{
    pulic void testOneNameValuePair( )
    {
        QueryStr query = new QueryStr("name = value");
        assertEqls(1, query.count( ));
        assertEqls("value", query.valueFor("name"));
    }
    public void testMultipleNameValuePairs( )
    {
        QueryStr query = new QueryStr('n1 = v1 & n2 = v2 & n3 = v3");
        assertEqls(3, query.count( ));
        assertEqls("val1", query.valueFor("name1"));
        assertEqls("val2", query.valueFor("name2"));
        assertEqls("val3", query.valueFor("name3"));
    }
```

```
public void testNoNameValuePairs( )
{
    QueryStr query = new QueryStr(" ");
    assertEqls(0, query.count( ));
}
public void testNull( )
{
    try
    {
        QueryStr query = new QueryStr(Null);
        fail("throws exception");
    }
    catch (NullPointerException e)
    {
    }
}
}
```

All the tests are passed by the code. But, it is complex. The methods count( ) and valueFor( ) have parsing code duplication. This duplication can be eliminated and can be passed in one place.

```
public class QueryStr
{
private str_query;
private QueryStr (Str queryStr)
{
if(queryStr = = null) throw new NullPointerException( );
– query = queryStr;
}
public int count( )
{
if(" ".equals(–query))
return 0;
Str[ ] pairs = – query.split("&");
return pairs.length;
}
public Str valueFor(Str name)
{
Str[ ] pairs = –query.split("&");
for(Str pair : pairs)
{
Str[ ] nameAndValue = pair.split("=");
```

```
if(nameAndValue[0].equals(name))
return nameAndValue[1];
}
throw new RuntimeException(name +"does not found");
}
}
```

It requires a single method that can do parsing and can eliminate the duplication. Instead of performing parsing of their own, the other methods works with results of parser. The constructor calls the parser that parse the data in the HashMap.

Introduce HashMap( ) into ValueFor( ) method. This make valueFor( ) method serve like parseQueyString( ) method which was required.

```
public string valueFor(str name)
{
HashMap < Str, Str> map = new HashMap <Str, Str>( );
Str[ ] pairs = – query.split("&");
for (Str pair : pairs)
{
Str[ ] nameAndValue = pair.split("=");
map.put(nameAndValue[0], nameAndValue[1]);
}
return map.get(name);
}
```

After refactoring, the programmers run all the tests. Extract method refactoring that is used to extract the parsing logic in its own method.

```
public string valueFor(Str_name)
{
HashMap<Str, Str> map = parseQueryStr( );
return map.get(name);
}
private HashMap<Str, Str> parseQueryStr( )
{
HashMap <Str, Str> map = new HashMap<Str, Str>( );
Str[ ] pairs = – query.split("&");
for(Str pair : pairs)
{
Str[ ] nameAndvalue = pair.split("=");
map.put(nameAndValue[0], nameAndValue[1]);
}
return map;
}
```

Again the tests are passed by using small steps. The changes can be controlled which minimizes the complexity.

Only valueFor( ) method can use parseQueryString() methods. To make parserQueryString( ) available to other methods, generate a _map instance variable and enable access to parseQueryString( ) method.

```
public class QueryStr
{
private Str_query;
private HashMap <Str, Str>_map = new HashMap<Str, Str>( );
public string valueFor(Str name)
{
HashMap<Str, Str> map = parseQueryStr( );
return map.get(name);
}
private HashMap<Str,Str> parseQueryStr( );
{
Str[ ] pairs = _query.split("&");
for(Str pair : paris)
{
Str[ ] nameAndValue = pair.split("=");
_map.put (nameAndValue[0], nameAndValue[1]);
}
return _map;
}
}
```

When the programmer switches from local variable to an instance variable, confusion can occur in operations. To avoid these confusions, use parseQueryStr( ) return_map.

```
public class QueryStr
{
private str_query;
private HashMap<Str, Str>_map = new HashMap<Str, Str>( );
public string valueFor(String name)
{
parseQueryStr( );
return_map.get(name);
}
private void parseQueryStr( )
{
str[ ] pairs = _querysplit("&");
for(Str pair : pairs)
{
Str[ ] nameAndvalue = pair.split("=");
_map.put(nameAndValue [0], nameAndValue[1]);
}
}
```

Again the tests are passed.

If parseQueryStr( ) is called before valueFor( ) method then the resultant would be as follows,

```
public class QueryStr
{
private str_query;
private HashMap<Str, Str>_map = new HashMap<str, str>( );
public QueryStr(Str queryStr)
{
if(queryStr = = null)
throw new NullPointException( );
_query = queryStr;
parseQueryStr( );
}
public string ValueFor(Str name)
{
return_map.get(name);
}
}
```

parseQueryStr( ) is called through a constructor.

Guard clause can be used to fix the problem successfully,

```
private void parseQueryStr( )
{
if(" ".equal(_query))
return;
str[ ] pairs = –query.split("&");
for(str pair : pairs)
{
str[ ] nameAndValue = pair.split("=");
_map.put(nameAndValue[0], nameAndValue[1]);
}
}
```

Due to duplication of parsing in count( ) method, it creates confusion. Instead of its own parsing use the _map variable.

Thus, it is changed from

```
public int count( )
{
if(" ".equals(_query));
return 0;
Str[ ] pairs = _query.split("&");
return pairs.length;
}
```

To

```
public int count( )
{
return _map.size( );
}
```

can delete this code if required.

The unparsed query string that is stored in _query instance variable is only required in parseQueryStr( ). It is reduced from an instance variable to a parameter parseQueryString( ).

```
public class QueryStr
{
private HashMap <Str, Str>_map = new HashMap<Str, Str>( );
public QueryStr(Str queryStr)
{
if(queryStr = = null)
throw new NullPointerException( );
parseQueryStr(query Str);
}
public int count( )
{
return_map.size( );
}
public string valueFor(Str name)
{
return_map.get(name);
}
private void parseQueryStr(Str query)
{
if(" ".equals(query))
return;
Str[ ] pairs = query.split("&");
for(Str pair : paris)
{
Str[ ] nameAndValue = pair.split("=");
_map.put(nameAndVlaue[0], nameAndValue[1]);
}
}
```

When the initial code and the above code is compared, they are quite similar. This change has occurred as a series of small refactorings.

5.5 INCREMENTAL DESIGN AND ARCHITECTURE

Q23. Explain incremental design with its working.

Answer : **Model Paper-II, Q10(b)**

Incremental Design and Architecture

In XP, during the development of an application, customers often change the current plan without acknowledging the developers and introduces new stories completely in every week. So, the programmers face challenging demand of completing 5 - 10 customer valued stories. But, as there is no advance preparation the developers finds it difficult to deliver the stories. Inorder to overcome this issue, the concept of incremental design was brought into practice.

Incremental design enables developers to build technical infrastructure in small steps so as to deliver the user stories in most efficient way.

Working

Initially a simple design element-basis method, class or architecture is created in such a way that it can solve the problems faced by the developers. Then, the design must be improved on regular basis depending on current requirements of software. The design levels in incremental design involves the concepts that are used in Test Driven Development (TDD). Each time the programmer creates a design, it must be with respect to the underlying problem.

A good programmer and well-experienced programmer thinks in several different perspectives. This concept allow programmer to create simplest and more efficient designs. When the programmer works with the design element for the second time, the design must be modified to make it general enough to solve two problems only. Then evaluate the design and make necessary changes so as to make the code clear and simple. When the programmer works with the design element for the third time, he should be able to solve the three problems at a time. Then, perform design review again and simplify the code.

Finally, when the programmer works the design element for the fourth and fifth time, the programmer will notice that its abstraction is perfect as per the customer requirements and this design will be more effective and simple.

Q24. Explain the following terms,

(i) Incrementally designing methods

(ii) Incrementally designing classes

(iii) Incrementally designing architecture.

Answer :

(i) Incrementally Designing Methods

This level of incremental design is similar to test driven development approach. When the driver implements the task the navigator focuses on the design. Navigator reviews each line of code and notes down the complex code and missing elements on the notecard. The navigators thinks about the necessary changes to be made for bringing improvements in the code and also decides which performing tests can be performed to drive the code into appropriate direction. During this process breakthroughs may occurs a number of times in an hour and refactorings occurs for every few minutes. It takes 10 minutes (or) more to complete the breakthroughs. This approach improves readability and maintainability which inturn results in development of a valuable product.

(ii) Incrementally Designing Classes

When the process of test driven development is performed in a proper way then the design of each class and method appears to be simple, elegant and easy to use. If the developers are not attentive while reviewing then the overall system design will become complicated.

The following are the factors that the navigator should focus on,

(a) Similarities between the existing code and other code.

(b) Clearly defined class responsibilities and concepts.

(c) Interaction between the existing class with other classes.

Whenever the programmers encounter a problem he/she note it down. Later they discuss the issue with the team members for achieving solutions and refactor it. If the programmer thinks performing modifications in design will affect the team members then they quickly discuss it with team members around the white board.

In a single day, refactorings at class -level can take place multiple times. Based on design, breakthroughs may occur multiple times per week and can be completed in several hours. In some situations, the programmer does not have much time to complete all refactorings that are identified. As long as the design is better than before the developers can perform n number of refactorings.

(iii) Incrementally Designing Architecture

The design of large programs uses all aspects of organizational structures known as architecture. For instance, most of the programs separates business logic classes, user interface classes and persistence classes into their own namespaces. This is referred as three tier architecture. These type of architectures are built by using recurrent patterns. They are standard conventions corresponding to code base. For instance, in a three-layer architecture every class of business logic may be a part of "business logic" namespaces, which may be adopted from a generic "business object" base class and probably interfaces with its persistence layer counterpart in a standard form.

The reoccurring of patterns represents the programmers application architecture. Though they generate efficient code, they are also a type of duplication which makes modification to their architecture more complex. The architectures can be designed incrementally. For growing software, the first choice of programmer should be making use of TDD and pair programming. The programmer must be attentive while introducing new architectural patterns. They must include only those features that are required at the moment.

In architecture breakthroughs can occur in every few months. Due to duplication, refactoring can take few weeks or more to support breakthrough. Once the programmer identifies the pattern of new architecture then it is not difficult . The programmers must select one specific part of their design and try to implement the new pattern. Observe for a week or more to ensure the change works effectively. When it is effective, their design new architecture and take the remaining system into action. The programmer uses slack in every iteration to correct other classes, programmer performs work every day and refactor every class that are used.

During refactoring, the programmer should deliver the stories recursively. If the programmer takes a gap from refactoring of new development then it will affect the customers. Technical excellence should be balanced with the value of delivering.

Q25. Explain about risk-driven architecture.

Answer : **Model Paper-I, Q10(b)**

Risk Driven Architecture

A risk driven architecture is a model that directs the software developers to design architecture by applying minimum set of architecture techniques that are enough to reduce their most critical risk. It prevents the developers from spending their time on low- impact techniques. The key element of risk driven architecture is to ensure that the risk can be identified clearly and does not mislead the developers. The following steps summarizes the risk-driven architecture,

(i) To diagnose and prioritize risks

(ii) To select and apply set of techniques

(iii) To evaluate risk reduction.

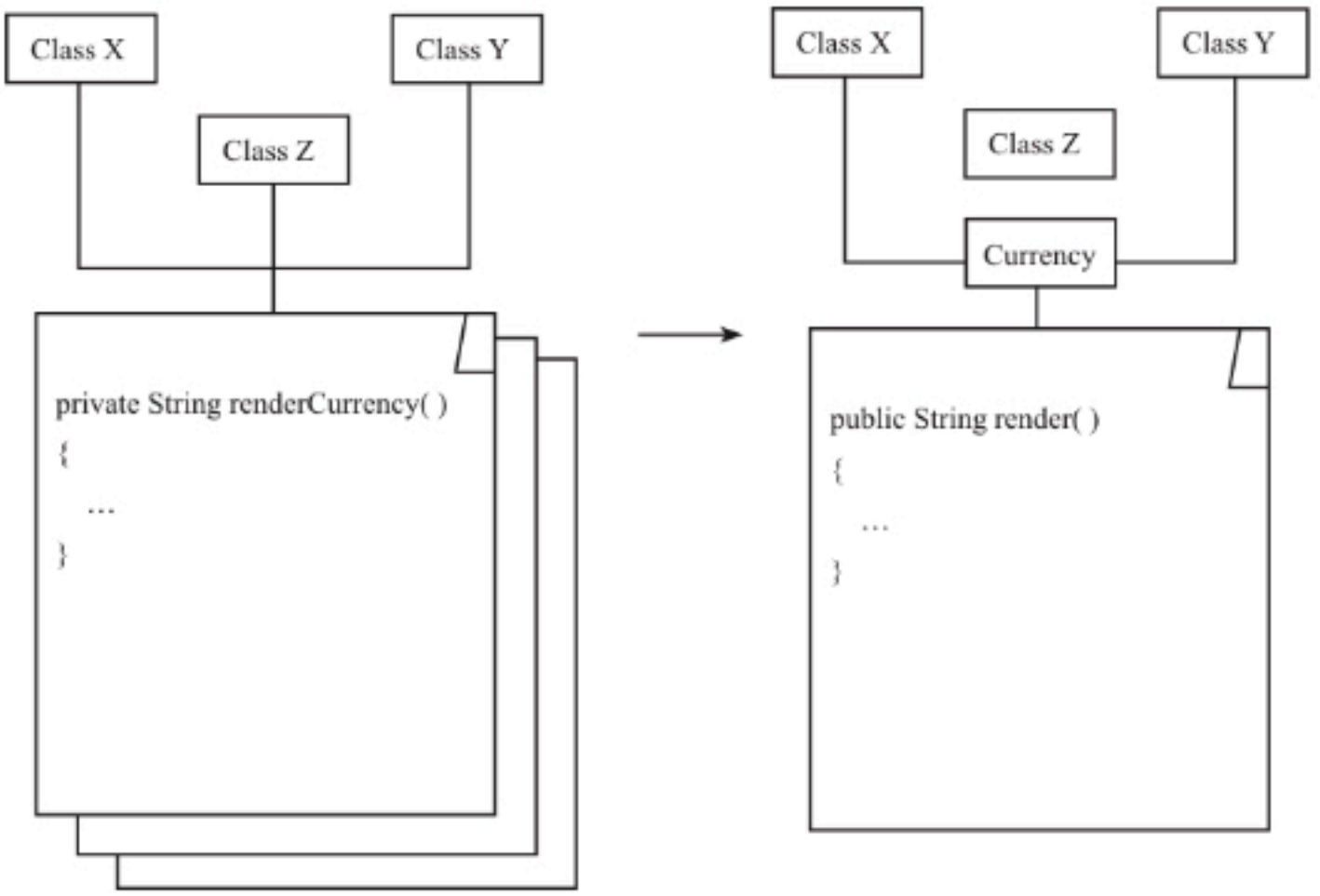
The programmer should not apply solutions to stories that are not scheduled. They are responsible to select the type of refactorings that they must work on.

It will be irrelevant, if the programmer applies the features that the customers are not looking for. The programmer can trigger the efforts of refactoring to decrease the risk. The programmer should make improvements to decrease the risk in future.

In implementation of Risk-Driven Architecture, the programmer should find the ways to eliminate duplication. The programmer should ensure that each and every concept has their own class. Then all formatting of every concept should be compressed in a single method in every class as shown in the diagram. If the programmer still faces duplication then the strategy pattern enables programmer to compress the formatting code for other work.

Look for the SIA GROUP LOGO on the TITLE COVER before you buy

The programmer should not support localized formats until and unless the customers request. The programmer should minimize the efforts to improve the existing design. When the programmer removes the duplications from the concept then it will simplify the structure of the code, increases maintainability and reduces technical debt over the time.



With Risk

The currency rendering algorithm is copied by each and every class. Modifying it would be complex and costly, if it is internationalized.

Without Risk

The currency rendering algorithm is applicable only to currency class. In this only one method needs modifications if it is internationalized.

The application of Risk Driven architecture provides the following advantages to its clients,

(i) It enables smooth communication between stakeholders and solution architect.

(ii) It ensures that project threatening risks are properly addressed by appropriate techniques.

(iii) It enchances the quality of solutions.

(iv) It reduces the risk of delayed delivery and budget overruns.

5.6 SPIKE SOLUTIONS

Q26. What are spike solutions? Explain.

Answer : **Model Paper-III, Q10(b)**

A spike solution is a small technical experiment that is used to explore solutions for a specific problem. It can be used in a number of situations which are as follows,

1. It helps the team in getting familiarized with new hardware or software.
2. It helps to evaluate a problem thoroughly and helps the development team to divide work among themselves individually.
3. It helps in reducing the risk occurrence in future and discover the issues/bugs that were left unnoticed in the previous iteration.

For instance, consider the following Java program to determine whether an exception is thrown on arithmetic overflow or not.

```
public class ArithmeticOverFlowSpike
{
    public static void main (String[ ] args)
    {
        try
        {
            int x = integer.MAX_VALUE + 1;
            System.out.println("No exception:x="+x);
        }
        catch (throwable e)
        {
        System.out.println("Exception:"+e);
        }
    }
}
```

Output

No exception:x = – 2147483648.

Implementation of Spike

Generally, a spike is implemented by creating a small program or test that describes some aspect of new functionality. It enables the programmer to also work practically not just theoretically to better understand the problem. Since writing code consumes more time when compared to reading, initially write small, standalone programs. Then execute the code either on test framework or command line and hard-code the values. The user input should be ignored, if it is not important. However, this approach does not allow the programmer to reuse the code in his/her actual production code base because it was not developed properly. When the programmer completes the experiment, it is reviewed, documented and shared with team members if needed.

5.7 PERFORMANCE OPTIMIZATION

Q27. Write about performance optimization.

Answer :

Performance Optimization

Performance optimization refers to the process of modifying existing system to improve its performance. However, this process is carried out only when it is necessary. One of the major challenges in performance optimization is to identify the actual reason/bug behind the problem. Sometimes, the persons working on the system fail to identify the bugs even after spending days on the same. In such situations, a fresh pair of eyes i.e., investigating it with a new person helps in the identification.

Performance optimization requires in depth understanding about the system behaviour. For example, understanding the behaviour of kernel, system bus, CPU, virtual file system, hard disk controller, cache, system memory etc., to identify how a file is read from disk. For this reason, it is difficult to pick the exact method which can directly improve system performance as it requires a detailed study on every aspect of the system. Modern developers use a brute force approach and predicts the system performance based on a small test program. This program is attached to the system code while running it for the first time. Such approach usually works well but sometimes leads to complexities that negatively effects system performance. Therefore, it is preferable to adopt the traditional approach of computing system performance. Based on the results, various parts of the system are modified and the performance is measured again. If the results are significant, the modification are accepted, otherwise other parts are considered for optimization.

The testing is usually performed using end-to-end test to obtain accurate results. However, testing tools like xUnit can also be used for this purpose. Make the test results report as pass or fail based on a specific criteria. In case of failure, the test is passed as input to the profiler where the bugs/errors are identified and their impact is reduced or eliminated. The test's are performed every time when a change is made to the code.

There are two major drawbacks associated with optimization. First, it increases the complexity of code and second, it consumes significant amount of time. For these reasons, it is performed only when the customer requires. The primary focus of programmers should be on generating a simple and efficient code. A test should be performed every time when there is a possibility of performance degradation.

Extreme programming uses a mechanism that offers prioritized customer needs. It considers release planning along with user stories to prioritize the optimization. However, the technical requirements of the programmers need to be described in terms of organizational perspective to the customers so that they can understand their importance. However, the programmers are responsible for keeping the development platform efficient.

Writing Performance Stories

Performance story is generated based on a specific customer requirement related to the software. These stories are usually in the following terms,

1. Throughput

It refers to the number of tasks to be completed in a specific time period.

2. Latency

It refers to the grace time or delay that is allowed for a task to complete.

3. Responsiveness

It refers to the time/delay that is allowed between the task and its associated feedback.

It is also necessary to decide the acceptable states for which the performance of the system is accepted. It must also provide the best possible state of performance. Both these states are important as the former one provides the information about reaching the minimum requirement whereas the latter one provides the information about stopping the task when it is reached.

Apart from all the above terms, stories are created by considering various other conditions like the platform on which the system will work, availability of bandwidth and latency, number of individual that use the system etc.

5.8 EXPLORATORY TESTING

Q28. Explain about exploratory testing and its tools.

Answer : Model Paper-III, Q11

Exploratory Testing

In XP, teams does not have individuals Quality Assurance (QA) department. There is no absolute group of people who are responsible for evaluating and ensuring quality assurance of final result. The entire XP team i.e., testers,programmers and customers bear sole responsibility for the result. In conventional teams, QA group was regularly rewarded for detecting bugs. Whereas, in XP there is no such motive program is organized for the developers for detecting or deleting bugs. The main motive of XP is not to write, detect or delete the bugs but rather it is to produce zero bugs.

Good testers will approach software with a new perspective like to search for breaks, holes and surprises. Testers help the team members to achieve their aim and improve their way of working. Exploratory testing provides an efficient way of identifying surprises, gaps and holes.

Exploratory testing is the process of simultaneous learning, test design and test execution. It helps the tester to detect holes, unexpected side effects, future performance and risk corresponding to quality of the product. Exploratory testing is the best complement to XP's raft of automated testing techniques.

About Exploratory Testing

Exploratory testing is identical to incremental design and test-driven development. Instead of designing a big collection of tests up-front, the tester should design test on their own, execute the test on the software and wait for results. Every test is a small experiment that discovers the limits and abilities of emerging software. The result of every test guides the tester to design the next test, thereby directing him to achieve ways that are unexpected.

Exploratory testing can be performed manually or with the help of automation. It's defining feature is not how to run the software instead it is strong feedback loop between test design, test execution and representation of results.

When the stories are "done done" or when the software is ready to be released then the exploratory testing works better. The tester don't have to test the stories that were completed in the present iteration. It is motivating when the response is quick. Sometimes stories are not "done done" even till the final day of iteration.

However, tester does not use exploratory testing to assure the quality rather it is used to give information about how the process of team assures quality.

The following are the four tools used by exploratory testers to explore the software.

Tool 1 - Charters

A charter is a one page simple document that is created at the initial phase of project development. It serves as a plan or blueprint to achieve the desired goal of the project. It consists of three quality attributes: stability, reliability and performance. Charters are similar to stories and they function well when written down.

Tool 2 - Observation

Exploratory testers are always attentive to determine for anything that is out of ordinary. For instance, the hard disk that starts accelerating when the software should not access the disk. These type of observations make exploratory testers ask questions, run more tests and explore in new ways. For instance, a tester may notice that a web-based system uses parameters in URL and thinks it is easy to edit URL. If the URL is the http://level.instance.com/edit?id=73, the tester replaces it as http://level.instance.com/edit?id=7777.

Tool 3 - Note Taking

Exploratory testers simultaneously note down all the actions that they perform while exploring. This prevents the tester from forgetting their activities. In addition, the testers also use screen recorders to keep track of activities carried by them.

This form of notetaking and recordings will show the work of testers that they did in every minute and hours.

Tool 4 - Heuristic

Heuristic is a technique that supports in decision making. The testers need not have to spend time in assessing all the potential options, conducting cost benefit analysis and ranking the alternatives, rather they can employ heuristics to arrive at the best option. The testers can improved the exploratory efforts by making and maintaining a list of essential heuristics when software exploring is performed. This same list can be used as a welcome reference for the programmers because they executes the software. Some of the heuristics will be specific to the tester's domain. For instance, if the tester works with software networking then the tester must work necessarily with IP addresses. The tester may test software to see how it responds to invalid IP addresses ("777.777.777.777"), special IP addresses ("198.0.1.1") and IPv6 style addresses ("::1/123"). Boundary testing is an example of a test heuristic.