Q4. How does continuous integration helps in reducing code collisions?

Answer : Model Paper-III, Q1(f)

Code collisions occur when there is a "mismatch" in logic, which lead to semantic errors. Continuous Integration demands shorter cycles of integration, which results in smaller changes in code and it would be much easy to trace the bugs when there is small change in code. The consequent result of such continuous integration strategy would be reduction in code collisions.

Q5. Define collective code ownership.

Answer :

Collective code ownership refers to a model wherein all members of the development team shares responsibility of maintaining the quality of code.

During a project development, it is a common practice to distribute various logical parts of the project to separate teams. Such logical distribution may include the development of graphical user interface, writing web services, code to interact with database, report generation, production of test data and so on.

Q6. What is multistage integration?

Answer : Model Paper-II, Q1(f)

Multistage integration is a software development technique which is used to achieve high degree of integration in parallel so as to reduce the risk of build failure. This technique mainly includes two separate builds - commit build and slower secondary build. In a commit build, all type of tests that are necessary for development of software i.e., unit tests, integration tests and end-to-end tests are included which runs synchronously. Whereas a slower secondary build includes performance tests, bad tests and stability tests and runs them asynchronously.

Q7. Write a short note on,

(a) Product documentation

(b) Hand-off documentation.

Answer : Model Paper-I, Q1(f)

(a) Product Documentation

This type of documentation is usually meant for the end-users. It includes user manuals, API references and examples, and various kinds of reports. This type of documentation caters business value to the end product, which is sometimes worthier than a very smoothly working product itself.

(b) Hand-off Documentation

This type of documentation is meant to "hand-off" or hand-over" the final product to another team. It includes detailed explanation with respect to various stages of evolvement of project. It also discusses various challenges faced during product development, reasons for those challenges, remedies adapted to overcome them, and ways in which those challenges may cripple the product.

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

UNIT 3

RELEASING

Marketed by:
SIA GROUP

PART-A

SHORT QUESTIONS WITH SOLUTIONS

Q1. What does the term 'done-done' refer to?

Answer : Model Paper-III, Q1(e)

The concept and meaning of the term 'done done' signifies that once the team completes the story, they should not come back to it. Typically, a 'done done' story is a data which is ready to deploy, it is free from errors, unintegrated and untested code.

Unfinished stories presents lot of hidden costs in projects. As a result, at the time of release the team had to complete unpredictable work. Thus, leading to disturbance in release planning and inhibiting the team in achieving the results.

Q2. What are the techniques used by XP to achieve zero bugs?

Answer : Model Paper-I, Q1(e)

The XP delivers myraid of techniques to achieve nearly zero bugs. They are,

(i) Write fewer bugs

(ii) Eliminate bug breading grounds

(iii) Fix bugs now

(iv) Test your process

(v) Fix your process.

Q3. List out the various types of version control systems.

Answer : Model Paper-II, Q1(e)

The different types of version control systems are as follows,

(i) Repository

(ii) Sand box

(iii) Check out

(iv) Update

(v) Lock

(vi) Revert

(vii) Tip or head

(viii) Tag or label

(ix) Roll back

(x) Branch

(xi) Merge.

PART-B
ESSAY QUESTIONS WITH SOLUTIONS

3.1 BUGFREE RELEASE

Q8. Write in detail the concept of releasing the code.

Answer : Model Paper-III, Q6

The process of releasing code emphasizes on the value of code and agile developer's value, i.e., designing software upon complete documentation. Therefore, inorder to achieve the results and make use of opportunities, the team must push the software to the production team as soon as possible. Some of the practices that helps the team to transform the big release push into 10 mins tap, done done, no bugs, version control, ten-minute generation, continuous integration, collective code ownership and post-hoc documentation.

So, to push the software into production, the concept of 'releasing' mini-etude is adopted. Basically, a novice user makes use of etude to design map which involves all steps to release software. These steps are useful to the team who are not skilled in XP. In particular, the etude not just modifies the process but also eliminates the communication bottlenecks. Inorder to make most efficient of etude, apply timeboxing for nearly half an hour every day.

This can be achieved by ensuring that the team uses the red, green, index cards, empty table or magnetic whiteboard for value stream map and writing implements. It includes the following steps,

Step 1

Initiate by creating dissimilar pairs like programmer working with customer, customer working with tester and so on so forth. This avoids the pairing by job description and also gives the designer opportunity to work with a new partner every day.

Step 2

This step is time boxed to atleast 10 mins. Here, every pair activity is bound to occur between the time who has idea corresponding to the process and the release time corresponding to the consumers.

Subsequently, every single iteration should be counted as one activity. But, if an activity consumes less time like less than a day then club those activities together to form one single activity. Also, include the waiting time of an activity.

The team can also choose any existing card if any new card is not found.

Continue to previous line the team should choose minimum of one task and write it on red card. All the changes should be reflected intermittently. The team should answer the question related to the time consumed in development once the software is released.

It is important to keep in mind that the team should not do guess work while estimating the time taken. They can make use of calender time. The card should contain three timings such as , shortest time, longest time and standard time required.

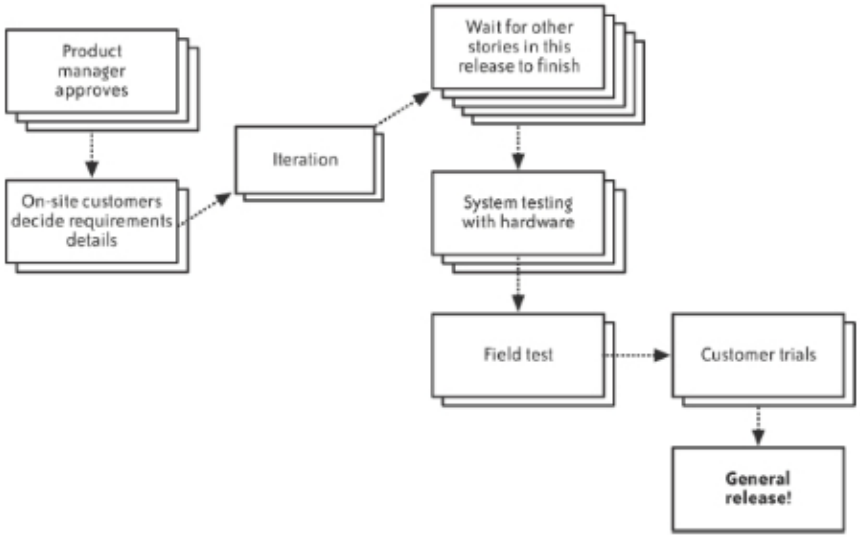
Step 3

This step is time-boxed to 10 mins. Here, the team should conduct meetings so as to minimize the time consumed by the activity to complete. Subsequently, select an idea and enter the data on the green card.

Step 4

This step is time-boxed to 15 mins. Here, the team reveals the cards on the table or whiteboard in terms of value stream map. Also, the data of those activities which occur first and last must be entered on the red cards.

The team can make use of white board to enter data such as drawing arrows in between cards so as make the flow more clear and place the green cards after red cards.



Figure

Consider following questions,

(i) Which step piles up the work?

(ii) Which results are unusual?

(iii) Which element acts as a constraint in the system and discuss the methods to enhance the system performance?

(iv) Which type of data on green card gives ideas to the team?

Q9. Explain in detail the production-ready software.

Answer :

The concept and meaning of the term 'done done' signifies that once the team completes the story, they should not come back to it. Typically, a 'done done' story is a data which is ready to deploy, it is free from errors, unitegrated and untested code.

Unfinished stories presents lot of hidden costs in projects. As a result, at the time of release the team had to complete unpredictable work. Thus, leading to disturbance in release planning and inhibiting the team in achieving the results. So, using the concept of 'done done' can help the team to meet the commitments at the end of every iteration. In doing so , the team can easily deploy the software in every iteration. It is important to note that how the software can reach 'done done' state. This is entirely based on the organization. If the customers can use the story once it is released it is said to be don - done. The list below illustrates the criteria which qualifies for the story completion,

(i) Perform all unit, integration, customer tests.

(ii) Develop the code

(iii) Design the refactored code to the level of satisfactory.

(iv) Combine the story from starting to the end i.e., UI to database which is compatible with the remaining software.

(v) Design the script and include the new modules.

(vi) Install the build script containing story in automated installer.

(vii) Migrate the script i.e., make necessary updations to the database scheme, once it is right the installer exports the data on different storage.

(viii) Carryout the process of review which includes reviewing the story and fulfilling the commitments.

(ix) Clear all the bugs/errors.

(x) Ensure that the story is accepted among the customers.

Other teams also focuses on documentation, performance and scalability.

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

Q10. Explain the following,

(i) How to be 'done done'

(ii) Making Time.

Answer :

(i) How to be "Done Done"?

The XP works efficiently, when the team starts progress in the software with respect to every aspect of it. This process is far better than saving last few days of iteration for achieving the state of 'Done Done' stories. More importantly, it not just minimizes the risk of incomplete work but also it presents easiness during the entire process.

With the use of test-driven development, developer must integrate testing, coding and designing. It is important to note that, if the team is working on engineering task, they should combine task with existing code. In doing so, continuous integration and keeping a time limit of nearly 10-mins is necessary. At the same time, also create engineering task to update and install. Inorder to avoid more problems, operate on only one pair of work in parallel with other.

Moreover, make on-site customers to participate in it and if it does not work then display the screen layout. Each time the customer sees it, they change UI. However, some work at last minute may come out, provided if the team delays any demos during the iteration. Once, all the software components are ready set them together and run it so as to ensure that all the components work together. Although, it is not testing but it comes under a good check before the release. Subsequently, take assistance from the testers so that they will introduce some exploratory techniques. Once these techniques are applied, the team can easily determine the errors, outright bugs and also will be able to fix them. So, this will help them improvise their code.

If the team is assured that the story has reached the stage of 'done - done' then introduce it to the customers for final acceptance review.

(ii) Making Time

When the amount of work increase and there is time limit then the team may feel impossible to achieve the commitment. However, the work can be accomplished by working on it throughout the iteration. And, the idea of saving the work up till the last days may create lot of problems. This step could be success if the length of the stories is short and it will be less tedious to complete it.

Moreover, the novice team members to XP, inevitably develop large stories which are too difficult to be 'done done'. During the development the team finishes the code but fails to finish the story. But, this contains lot of bugs. The teams decide the schedule as well as stories to be signup. In doing so, if the stories are large then break it into multiple components and operate on it one at a time in every iteration.

When the team faces problem in getting the stories done - done, then they should ignore those stories that effect the velocity. Count that story with minor UI bugs as zero at the time of calculating the velocity.

As a consequence, this makes the velocity less thereby making the work more manageable in next iteration. This lower velocity makes the team to schedule two stories in single iteration. So, it implies the stories are too large to schedule, but it should be splitted and then work on it.

Q11. Explain how it is possible to release software without dedicated testing.

Answer : **Model Paper-I, Q6(a)**

The team develops code using C and a bit of assembly language. Intentionally, develop the code erroneous by adding concurrent programming. Also, use Java multithreading library as it provides lot of bugs. To make it more difficult, the team makes use of a problem domain which is real-time embedded systems. Moreover, to operate on these type of programming, they hire novice programmer.

After operating on this, the novice team may consume more than the estimated time. The productivity is guaged minimum of three times in comparison to embedded software teams. In particular, the first field test units consume nearly 6 months for development. After this, other engineering disciplines are implemented to improvise the software.

Not every code developed by novice programmers is free from bugs (no bug). However, under this situation, the XP team can drastically minimize the rate of bugs. This comes upto average of half bugs per month in this difficult domain. With the independent analysis, the company operating a variant of XP reduced the bugs from thousand to hundred. However, the team achieves this cost-effectively and delivers above-average productivity.

Therefore, these novice programmer are then trained by XP experienced developers, so that the work commitment in adapting XP practices can be done easily.

Q12. How to achieve nearly zero bugs?

Answer : **Model Paper-II, Q6**

There exist numerous approaches to improvise the quality of software. This includes searching, eliminating the defects using conventional methods like testing, inspection and automatic analysis. Here the afford is to eliminate the defects completely and agile approach helps to generate fewer defects. The XP delivers myraid of techniques to achieve nearly zero bugs. They are,

(i) Write fewer bugs

(ii) Eliminate bug breading grounds

(iii) Fix bugs now

(iv) Test your process

(v) Fix your process.

(i) Write Fewer Bugs

When the programmer finds bugs in the coding, they can implement an approach to software development. This approach is test driven development (TDD). This technique is meant to minimize the total number of errors that the team generate. Once, this is done, it entails an extensive set of unit and integration test. Additionally, it concise the work into small, simple and veritable steps. So, if the team uses TDD reports then, they don't need any debugger.

Inorder to improve the advantages of test-driven development and work sensible hours, then program the production code in pair-wise manner. As a consequence of the pairs the teams brain power starts getting better which inturn assists in committing less errors and also helps to identify errors quickly. Apart from this, it also facilitates positive peer-pressure which is required to maintain the self-discipline in defect reduction process.

The use of test-driven development is also for eliminating coding defects. To avoid good code operating errorneously which are nothing but requirement oriented defect start working with stakeholders. More importantly, make the customers to sit with the team so as to conduct the meeting to discuss complex domain rules. Also, make use of customer test to fill the gaps in the requirements. Then, it is mandatory to demonstrate the software to stakeholders weekly.

(ii) Eliminate Bug Breeding Grounds

The occurrence of bugs in the coding is common but writing les number of bugs is foremost step in minimizing the total defects. So, once this is done it is considered that the team is ahead of other teams.

Despite of the implementation of test-driven development, the code contains technical debt. These debt (bugs) will be present usually in design and make the code susceptible to defects and unchangeable. As a result, it accumulate in certain part of the system. It is said that, the team cannot avoid these flaws. There exists some uncertainties wherein the team experiences, a design which appears to be good at first and then fail subsequently. Sometimes, the given shortcut with acceptable compromise will create lots of bugs, at times the requirements changes which leads to the change in the design.

So, it is assured that technical debts presents complication and ambiguity. These problems can be addressed by making small improvements week by week.

(iii) Fix Bugs Now

The errorneous code containing lost of bugs is presents high cost while fixing it. Subsequently, those bugs which are unattended and unfixed are prone to create more problems. Every single bug is caused due to the flaws present in the system and they keep increasing until they are fixed. Once, they are fixed, they result in the improvement of quality as well as productivity.

The process of bugs fixation initiates by designing an automated test which demonstrates the bug. Based upon the defect, the team applies unit test, integration test or customer test upon reaching the failing test, the bug can be cured.

After fixing the problem, still the main question is left which says what is the underlying cause of the bug occurrence. So, to overcome this, set up a meeting to discuss this with pairing partners. Then brainstorm the reasons for it to occur such as was there any design flow, Is it possible to modify the API to generate more of these bugs, what could be the method to refactor the code which does not generate this type of bugs.

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

The different types of version control systems are as follows,

(i) Repository
(ii) Sand box
(iii) Check out
(iv) Update
(v) Lock
(vi) Check in or commit
(vii) Revert
(viii) Tip or head
(ix) Tag or label
(x) Roll back
(xi) Branch
(xii) Merge.

(i) Repository

It is the master storage where the developer can store all files and history. Ideally, its is archived on version control server. Subsequently, every single project operating on its own must have there own repository.

(ii) Sand Box

It is also referred to as working copy. It can be described as something in which the team operates on the local development machines.

The role of the box is to store copies of files in the repository. The process of storage is taken from a specific point of time.

(iii) Check Out

It is important to check out the copy of the repository provided if the programmer has to create a sand box. It means 'update and lock'.

(iv) Update

It is used to update sand box so as to acquire latest changes from the repository. It is also possible to revert the status of the system to the previous state.

(v) Lock

It is helpful to perform editing in the file. It gives access to the intended team only but not the others.

(vi) Check in or Commit

It is used to check in the files into the system's sand box so as to store it in the repository.

(vii) Revert

It is used to restore the sand box to the original state. This is necessary to throw away the modifications made. The process of revert appears to be faster than the process of debugging.

(viii) Tip or Head

It contains the latest updates or changes. Each time the sand box is updated, the files are available at tip.

(ix) Tag or Label

It represents a specific time in the history of repository. It gives multiple access.

(x) Roll Back

It is used to eliminate the check in from the tip of the repository. However, the mechanisms for carrying out. This activity differ based on the version of control system used.

(xi) Branch

It appears when the programmer divides the repository into different 'alternate histories'. This process is referred to as branching. Each branch contains files, these files can be edited in a single branch independently of other branches.

(xii) Merge

It is a process of combining one or many changes. It doing so, it also solves the conflicts occurring in the process.

Suppose, if two programmers modify the file separately, then the second programmer had to merge its files with the first programmer.

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

So, once the systemic problem is determined, it can be discussed with the remaining team members. The process of fixing the bug demands the involvement of entire team. They implement the concept of collective code ownership where in any pair is utilized to fix the module full of bugs. Subsequently, the customers and testers also raise their personal experiences and problems while dealing with the code. Inorder to solve the old and new issues and bugs in the code the whole team discusses it and solves it.

However, in practical case it is impossible to fix every bug right a way. Sometimes, the bug arises in the middle while working. For this type of issue, the programmer handles it 10 or 20 mins later or asks the navigator to write problem on to-do-list.

(iv) Test Your Process

When these tests are applied on the code, the number of bugs can be minimized to large extent. These bugs which are minimized are those which cause lot of problems. So, the programmer prior to developing the code should estimate the problem domain.

Experts define exploratory testing as an invaluable way to widen the understanding of various types of problem that are likely to occur. Based upon the experience and instinct, the exploratory tests can determine what type of bugs are most problematic. Typically, the exploratory testing is effective in identifying the unexpected bugs. Due to its efficient feature, the team entirely relies on it and don't do their work. So, experts say that, it could be bad to depend upon such type of testing for identifying the bugs. It is important that they should only follow test-driven development and other good practices.

More importantly, use exploratory testing to test the process. The moment the test identifies bug, it indicates the developers work habits, process. At first the bug and then fix the process.

An efficient exploratory tester generally determines many bugs and fix the bug so as to fix the process.

(v) Fix Your Process

The bugs represents anonymous flaw in the implemented approach. Once, the total count of bugs go low, the process eventually becomes correct. The processes proceeds like this-once the bug is found, initiate the designing of the automated test and improvise the design so as to make the occurrence of bug less. This step is referred to as root-cause analysis. As the process moves further designing test and fixing the design is done. So, certain type of questions arise such as why there is not any test that inhabits the growth of bug?

- ❖ Why should a design need to be fixed?
- ❖ Make use of "five whys" so as to analyse the root cause.

After this, conduct a meeting wherein possible work habits can be discussed to improvise the code.

3.2 VERSION CONTROL

Q13. What is version control? Discuss its types.

Answer : Model Paper-I, Q6(b)

The programmers can work as a team so as to coordinate the source code, carryout test and other necessary project artifacts. To fulfill this need, the concept of version control system is introduced. It provides central repository which assist in coordinating the changes to the files and also providing the list of changes.

If the project does not have version control then, it contains snippets of code spread across the developer's machine, net worked drives and removable media.

When the project having version control make use of version control system to convey the modifications, this process is streamlined one and the developers acquire latest code from the server. They perform their work, run all the list so as to affirm that the code functions appropriately and after this verify the modifications. This entire process is referred to as continuous integration. This occurs multiple times a day with respect to each pair.

Q16. Discuss the terms,

(i) Automate your build and how to do it.

(ii) When to automate.

Answer :

(i) Automate your Build and How to do it

The programmers task becomes very easy and convenient if they get the opportunity to build and test the complete product. This includes creation and deployment of the package at any moment of time.

However, it is known fact that the task of producing a build is a time consuming and a frustrating job. Such situation cn create lot of troubles like the frustration can be contagious and other team members can get effected with it. This eventually effects the work and the team may enter the state of dilemma and start anticipating certain questions like,

1. "Will the team be able to release the software in few days only"?
2. "Will the developed software operates"?
3. "Will the team can develop everything"?
4. "Will the team be able to make the software demo to stake holders"?
5. "Will the team be able to fulfill the commitments at the time of emergency"?

Mostly, the build automation can be done quite easily and it is ignored. So, if any team which has not built it yet, then they should start working on it now.

Using tools like java, Ant,.IN.,NET, NANT, and MSBuild, the automate build can be developed. Platforms like perl, python and ruby contain their ohm build tools.

Ideally, the build should be complete but not complex. The tasks that it should accomplish are code compilation and running the tests, configuring the registry settings, initializing the database schemas, establishing the web servers, launching the process. So, it should do this without human involvement i.e., from scratch till testing the software. Once this is done, deploy it to the servers adding the production release.

(ii) When to Automate

In the inception (beginning) of the project and the first iteration, the team should establish the bare-bones build system. The role of this iteration is to generate a product which can be used in entire system. In doing so, the team should deliver the working product to the stakeholders.

Usually, the products are small in size and it is easy for the programmers to develop it. They produce high-quality automated build. It should contain build, test and deployment features such that with every iteration the build script get better. Typically, the build script is used to perform process of integration. Inorder to ensure that the build is updated, it is important that the team should not configure the integration machine manually. However, by doing the modification to the build script it can be configured.

Q17. Briefly write about the concept of ten minutes or less.

Answer :

Using the concept of great build, the teams can perform a way better than other teams. However, in making the whole system ready to use at any time, the team fails to notice that there rises a new issue related to it which is the build becomes very slow.

Moreover, by implementing the continuous integration, the team starts integrating in every few hours. This integration is divided into two builds one runs on team machine and other runs on the integration machine. Subsequently, inorder to continue normal processing, it is important to ensure that both the builds are completed. This is because, the team should not build break in XP. Once it breaks the changes has to be rolled back.

Typically, a build of 10 minutes entails a 20 minute integration cycle. Through, analysis it is concluded that, the delay is very long. And mostly, the team accepts 10 or 15 min of integration cycle. On the other hand, if the build time is kept under minutes from the starting itself then time can be easily controlled. Suppose, if the novice XP team exceeds the build too long, then it can be fixed similar to the process of fixing technical debt. In essence, it can be treated component by component and ensuring that, progress is made at each step. Many teams feel that, the tests they perform serves as a source for slow build. This is because, they lack focus.

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

3.4 CONTINUOUS INTEGRATION

Q18. Explain the need for continuous integration and how does it helps in reducing code collisions?

Answer : Model Paper-II, Q7(a)

The ultimate aim of continuous integration is to be able to deliver the software "at all times". In other words, the developer must not push activities till the final release. This approach maintains code of each member integrated and develops release infrastructure along with the rest of the application.

Generally, the activities which are pushed till final release include merging individual pieces of software into final release preparation of documentation and manual creating installer prepopulation of database and so on. These activities need their own time and they equally serve as any other regular activity like code development.

Delaying these activities may lead to leaving the final release with poor installer script, lack of documentation or even introduction of new bugs during integration of software into final release.

Code collisions occur when there is a "mismatch" in logic, which lead to semantic errors.

Continuous Integration demands shorter cycles of integration, which results in smaller changes in code and it would be much easy to trace the bugs when there is small change in code. The consequent result of such continuous integration strategy would be reduction in code collisions.

Q19. Discuss few practical strategies to employ continuous integration in the project work.

Answer :

In order to keep the code ready to release at any time, the members of project may adapt the following two habits,

1. By integrating of code pieces regularly say, after every 3 hours or 6 hours.
2. By keeping the build and related tests up-to-date. This includes installing script and infrastructure also.

Hence, after every major change in the code, prepare the new build, test it and commit the change to repository for new release.

The aim of such frequent builds is to keep the code always ready to release, or so close to final release that it need not require more than few minutes or an hour.

Q20. Discuss the problems and disasters of breaking the code building practice.

Answer : Model Paper-III, Q7(a)

Delaying and ignoring the practices of "building" may consume hours or even days of chasing the bug without success. Another effect of ignoring the "builds" is production of "ugly code", due to the large gaps of time between builds. But, with large gaps between builds it is not always possible to correct the bug "in place", because the effect of change in code to correct the bug may reciprocate as large number of changes in several other parts of code. To overcome these difficulties developers are forced to produce "ugly code" in which variable names, function calls, parameter values and other related data must be duplicated or aliased or modified in a non-standard way.

On the contrary, practice of consistent, regular and frequent builds reduces the scope of bug to the only code which was added after the previous successful build. Developer would be sure that bug is confined to small changed code.

Q21. Explain, in detail the role of continuous integration script.

Answer :

For a smooth progress of project development each of the software teams must ensure that a code that works on developer's machine must also work on any other machine, and a code that fails to work on a developers machine should not be accessible to any other team.

Inorder to achieve the above results, the best approach is to dedicate one machine as "integration machine". Integration machine is accessible to each team. But, before accessing the integration machine the team must ensure that "integration token" is accessible to it. (Integration token is some physical item which is kept beside integration machine). If the integration token is not available beside the integration machine then it means that another team is using the integration machine to integrate it's own code into the repository and other teams must wait till that team integrates its new successful build.

After a successful integration the team replaces back the integration token beside the integration machine. But, in case the team fails to make a successful build then it must not integrate it's new build before it puts back the integration token.

In a networked environment, the whole process can be automated with a script which can immitate the role of integration machine as well as integration token.

Q22. How do you deal with slow builds during continuous integration? Discuss in brief about multistage integration.

Answer :

The task of practicing continuous integration with slow builds is the most common issue faced by the software development teams. The programmers must try their level best to keep their build under 10 minutes. If the build is too slow, then synchronous integration becomes unsupportive resulting in more difficult integration. In order to overcome this issue, team must employ asynchronous integration wherein the members need not wait for integration to finish and can move on to their next task.

One of the major disadvantage of asynchronous integration is, it results in broken builds. When the build breaks and if other paired member tries to pull that code that is in source control then even his process fails. Besides this, if the pair who has broken the build is not available and mean while if some other member check in that code then he has to fix that build. This may increase the cycle time. But, while dealing with slow builds use of asynchronous integration is the only solution. So, the team must try to improve speed of their build by frequently integrating their work until the build time gets down to a reasonable number 15 or 20 minutes and then switch to synchronous integration.

Multistage integration is a software development technique which is used to achieve high degree of integration in parallel so as to reduce the risk of build failure. This technique mainly includes two separate builds - commit build and slower secondary build. In a commit build, all type of tests that are necessary for development of software i.e., unit tests, integration tests and end-to-end tests are included which runs synchronously. Whereas a slower secondary build includes performance tests, bad tests and stability tests and runs them asynchronously.

Although multistage integration is useful in teams that perform sophisticated testing, it is mostly employed for a slow test suite. This technique provides developers with instant feedback about the problems and helps them in maintaining the codebase free from test and build failures.

3.5 COLLECTIVE OWNERSHIP

Q23. What do you infer from the phase "collective code ownership"? Explain the concept in your own words.

Answer : **Model Paper-I, Q7(b)**

Collective code ownership refers to a model wherein all members of the development team shares responsibility of maintaining the quality of code. During a project development, it is a common practice to distribute various logical parts of the project to separate teams. Such logical distribution may include the development of graphical user interface, writing web services, code to interact with database, report generation, production of test data and so on.

Consequently, each team works on its assigned task and leaves the assembling of these parts of project to the project manager (and his/her team). One team may not be concerned with logic or code used by another team and so they does not take responsibility of another team's work.

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

Q14. Write short notes on,

(i) Time travel

(ii) Keep it clean

(iii) Single codebase.

Answer :

(i) Time Travel

The version control system offers a characteristics feature, which allows the programmer to go back in time. In essence, the programmer can easily update the status of the sand box from any specific point in past. In doing so, the programmer utilizes different debugging. Each time, the programmer encounters a challenging bug and is difficult to debug then go back to the old versions of code and set the system to previous timings where the bug did not exist.

Once, this is done move back and forth inorder to isolate the correct check-in which introduced the bug. Subsequently, the programmer can initiate the review process for checking the changes in check-in alone and also to know the cause of occurrence of bug. Now, the programmer performs integration consistently so as to minimize the total number of changes.

On the other hand, the time travelled is also utilized for generating bugs. If a bug is reported and it cannot be reproduced then utilized the same versions of the code which the users reported. However, if the bug cannot be fixed in current version and can be fixed in older version then using unit test that bug can be fixed.

(ii) Keep it Clean

The concept 'keep it clean' emphasize that the programmer should keep the code clean and should be ready to use or ship. It is an important concept in XP. Typically, initiate the process using sand box and the progress can be made after breaking the build in the sandbox. Also, it is important that the programmer never check in code which breaks the build. As a consequence, any member can carryout the task of updation without worrying about breaking the build. This can allow the team to work smoothly and share easily.

Moreover, the build generates the release automatically. However, the code might be clean but the software cannot be ready for the use. Some of the elements affected here are stories will be incomplete, many elements corresponding to user interface will be missing and some components do not work at all.

At the end of every single iteration all the loose ends will be finished and all the stories will reach the 'done done' state. And after this give a demo to the stakeholders. The generated software shows a genuine increment of value corresponding to the organization. Subsequently, it should be returned by tagging the tip of the repository.

(iii) Single Codebase

The team can start making duplicate copies of the code base which can cause more troubles. This initiates when customer requests a customized version of software. In doing so, they intend to deliver it quickly and make duplicate copies of codebase. Minor changes are needed and then it is delivered to the customers.It simply doubles the number of lines.

This extremely, deteriorates the capabilities of working software corresponding to the time. Also, it is not possible to integrate the duplicated codebase without the implementation of quick action. A single click can lead to technical debt. And through version control system, the customer makes lot of such mistakes. There is also an option of branching the code. It involves the division of repository into two individual lines of development. This is nothing but duplicating the codebase.

3.3 FAST BUILD

Q15. Discuss the concept of ten-minute build.

Answer : Model Paper-I, Q7(a)

When an employee joins a new company, he is given an hour to new work space. This is time consuming because the company has many build tools, libraries in version control. It is efficient version control system as it decreases the modifications and the process becomes very slow. Storing tools and libraries in version control enables the data updation.

Once the tour is finished, the new employee goes to the root of source tree and type build. Now, he compiles and checks as the build information proceeds. This includes more than just building source. It also contains complex application which needs web server, one or many web services and several databases. On the other hand, the new employee spends initial days in configuring the workstation. When it comes to test environments, the company has to suffer with lot of problems because while dealing with problems, the team has to stay idle for many days. Inorder to ease this, environment sharing can be done but this also presents problem as two tests cannot be executed simultaneously.

Furthermore, the step is automated. This implies, it allows the programmers to run tests on their own machines irrespective of time. Eventhough, it is disconnected from network, the process of building still continues. Its role is to configure the local web-server, initializing a local database.

This approach may seem logical initially but some critical situations like, team lead falling risk, a team member attending an emergency holiday, a critical team member suddenly retires, a team member stays home with a sick family member, etc., may leave the team handicapped depending on the role played by the missing member of the team.

On the contrary, collective code ownership promotes the culture of sharing and taking part in tasks assigned to other teams. Each team should try to understand the code developed by another teams, so that the critical situations of missing team members does not paralyze the team's task or the whole project. Improved code quality is the key component of collective code ownership.

Collective code ownership is a phrase coined to develop the thought in minds of project members that tasks assigned to each team are "not confined" to the concerned team, but "just concentrated" over it, with respect to responsibility. In other words, as the code base is owned by the entire team, any member in the team can make any modifications anywhere.

Q24. Suggest a possible approach to make collective code ownership work.

Answer : **Model Paper-III, Q7(b)**

One of the possible approaches to make collective code ownership work is to strongly and repeatedly suggest the project members to agree on producing "good code". When such a thought roots deep into project members minds a natural consequence and effect would be "to improve, or correct, any error or deficiency found in the code, irrespective of whether the code belongs to them or other team members. In collective ownership, a joint commitment is required from the team members so as to develop efficient code. The programmers must try their best to develop code of high quality.

This approach also has a secondary benefit i.e., no one needs to be perfect. Each team member feels comfortable with the thought that there is always "help" available for his/her code's improvement. It boosts the morale of all project members which further enhances the spirit of collective code ownership.

Q25. In a project work, when adapting collective code ownership strategy, how do you deal with a programming code which is unfamiliar to you and does not fall into your immediate programming skills?

Answer :

Most feasible way to enjoy the benefits of collective code ownership is to "pair up" with a local expert for any code that you do not understand or is unfamiliar to you. Discuss with him/her few parts of unfamiliar code and use his/her explanation to infer the code workings.

To get even better knowledge of that unfamiliar code it would help to refactor the code. Refactoring of code always leads to simpler logic.

Another tool which may help to understand some unfamiliar code is abundant testing of the code using various test data. Changes in test data leads to various kinds of test results and so there is high possibility that it leads to understanding the logic used in the unfamiliar code.

Q26. Mention some of the hidden or obscure benefits of adapting collective code ownership.

Answer :

One of the obvious benefits of collective code ownership is that whole codebase is made available to the developer and hence they can modify and improve any part of the code at anytime. In addition to this obvious benefit, there are several other hidden benefits of collective code ownership.

For example, if a team member is assigned with the task of database management but if he/she also enjoys writing their own code then they can join the team of code development and write new code or improve their code by making necessary changes in the code. Moreover, the developers need not carry the burden of maintenance of code written by them because as they have already shared the logic and skills with other project members, so there are several other hands and heads which can deal with their code.

3.6 DOCUMENTATION

Q27. What is the purpose of documentation? Broadly, categorize various types of documentation produced or required during and after a projects completion.

Answer : **Model Paper-II, Q7(b)**

The main purpose of documentation is "communication". In a project development, several "parties" take part which includes project members, clients and future customers or programmers. Documentation is communication between each pair of these parties. Documentation helps project members to get better grip over specific parts of projects, to describe the specifications of the project to the end-users or clients, or to detail the sketch of inner workings of various parts of the project, so that the project work can be extended by future programmers.

Documentation can be broadly categorized into three types which are as follows,

(i) Work-in-Progress Documentation

This type of documentation is meant for the project members to understand various parts of project, their role in each part of the project and interaction between these parts of the project.

This documentation need not be compiled in written form as booklets, but instead it may include the conceptual designs in the form of flowcharts, clear sketches, notes on oral explanation, or detailed stories in the form of wiki or spreadsheet calculations or recorded audios and videos.

(ii) Product Documentation

This type of documentation is usually meant for the end-users. It includes user manuals, API references and examples, and various kinds of reports. This type of documentation caters business value to the end product, which is sometimes worthier than a very smoothly working product itself.

(iii) Hand-off Documentation

This type of documentation is meant to "hand-off" or hand-over" the final product to another team. It includes detailed explanation with respect to various stages of evolvement of project. It also discusses various challenges faced during product development, reasons for those challenges, remedies adapted to overcome them, and ways in which those challenges may cripple the product.

The ultimate aim of hands-off documentation is to convey necessary knowledge acquired during product development, to the new teams, so that the product sustains and exists in maintainable state.

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