Q.1 (a) Pair programming has become a popular XP tool because code is regularly reviewed by more than one person, resulting in higher quality code. Identify and describe the roles and responsibilities in the Pair Programming process?

- Pair programming is a method of programming in which two people work together at one keyboard.
- One person, "the driver", types at the keyboard.
- The other person, "the observer" (or "navigator") reviews each line of code as it is typed, checking for errors and thinking about the overall design.

```
Driver:
            -Write the code according to the navigator's specification
            -Listen intently to the navigators instructions
            -Ask questions wherever there is a lack of clarity
            -Offer alternative solutions if you disagree with the navigator
            -Where there is disagreement, defer to the navigator. If their idea fails, get to
          failure quickly and move on
            -Make sure code is clean
            -Own the computer / keyboard
            -Ignore larger issues and focus on the task at hand
            -Trust the navigator - ultimately the navigator has the final say in what is
          written
            -You are writing the code

          Navigator:
            -Dictates the code that is to be written - the 'what'
            -Clearly communicates what code to write
            -Explains 'why' they have chosen the particular solution to this problem
            -Check for syntax / type errors as the Driver drives
            -Be the safety net for the driver
            -Make sure that the driver sticks to the small task at hand
            -Outline and note down high level tasks / issues
            -Ongoing code review
            -Pay attention
            -Wait until the task is complete to bring up design / refactoring issues

          Both:
            -Actively take part in programming
            -Aim for optimal flow - avoid trying to be 'right'
            -Embrace your role
            -Intervene if your pair is quiet
            -Know when to give up / steal keyboard
            -Communicate, communicate, COMMUNICATE!
            -Sync up frequently to make sure you are on the same page
            -Don't hog the keyboard
            -High-five every time a test passes
            -Follow best practices for TDD
            -Swap roles frequently
```

(b) Discuss the benefits of Pair Programming that contribute to the development and quality of the product.

- Some benefits you can expect:
  - better code (simpler design, fewer bugs, more maintainable),
  - higher morale (more fun!),
  - shared knowledge throughout your team (both specific knowledge of your codebase and general programming knowledge),
  - better time management,
  - higher productivity.
- Pair programming offers several important benefits that contribute to the quality of the product.
- The first is that both programmers are learning from each other and becoming more familiar with different aspects of the product. This is important for continual improvement and cross-training, but also helps expedite code defects as they arise during the build.
- The second benefit is that pair programming allows each developer to focus on his or her specific role; there is a lot less cognitive overload when a developer can concentrate on getting the code working while letting the pairing partner focus on things such as performance optimization or possible bugs.
- Finally, pairing can also build trust and encourage more regular communication between team members.

Q.2 (a) In Scrum, differentiate between the different roles and responsibilities of the Product Owner, the Scrum Master, and the Team.

Product Owner

- The product owner selects work requirements for each sprint.
- The product owner maintains a prioritized list of requirements for the product called a **product backlog**.
- The requirements are written in the form of **user stories** or stories about the problem that needs to be solved by the requirement.

Duties of the Product Owner:

- PO has to attend the daily sprint planning meetings.
- PO prioritizes the product features, so the development team can clearly understand them.
- PO decides the deadlines for the project.
- PO determines the release date and contents.
- PO manages and creates the product backlog for implementation, which is nothing but the prioritized backlog of user stories.
- PO defines user stories to the development team.
- Spending some time to prioritize the user stories with few team members

Scrum Master

- Tracking (progress) is also an important part of the Scrum approach.
- Scrum teams meet daily in what is commonly referred to as a **daily stand-up** or **daily Scrum meeting**.
- A **Scrum master** is assigned to lead these meetings to ensure that they remain brief and focused and that all team members have a chance to contribute.

Duties of Scrum Master:

- SM facilitates team for better vision and always tries to improve the efficiency of the teams.
- SM manages Scrum processes in Agile methodology.
- SM removes impediments for the Scrum team.
- SM arranges daily quick stand-up meetings to ensure proper use of processes.
- SM helps product owner to prepare good product backlog and sets it for the next sprint.
- Conducting retrospective meetings.
- SM organizes and facilitates the sprint planning meeting.

Team

- These daily scrum meetings are designed to be short (usually around 15 minutes), and each team member answers the following questions:
  - What have I done since yesterday?
  - What am I doing today?
  - Any roadblocks?
  - The team also uses **burn down** or **burn up** charts to track progress through the sprint.

Duties of Team

- Create and deliver the products or services;
- Be self-organized and self-managed. The Teams must be able to determine its own tasks and how they will perform it;
- Cross-functional. Teams are not combined with a single skill, but multiple different skills;
- Are dedicated for a product or a service.
- Recommended to work in the same work space.

(b) In Scrum, what is the purpose of the Working Agreement? What issues or topics are addressed by the Working Agreement?

- One way that trust and teamwork are established and enforced is through the creation of a working agreement.
- This is a document or set of expectations that define how the Scrum team is going to work together.
- The working agreement is the first point of collaboration for a new Scrum team as they define their relationships.
- It is more than just rules of engagement for team behavior; the working agreement ultimately reflects the values and commitment of the team.

Work agreements are the set of rules/disciplines/processes the team agrees to follow without fail to make themselves more efficient and successful.

The purpose of the **working agreement** is to ensure the Agile Team shares responsibility in defining expectations for how they will function together and enhance their self-organization process. It creates an awareness of both positive (and negative) behaviors that can impact the Team and empowers the Scrum Master to keep them accountable.

Topics :

- Time and location of Daily Scrum
- Testing strategy (unit, functional, integration, performance, stress, etc...)
- Build and infrastructure plans (shared responsibilities)
- Team norms (be on time, respect estimates, help when needed, etc...)
- How to address bugs/fires during Sprint
- Product Owner availability (phone, office hours, attendance in Daily Scrum)
- Capacity plan for initial Sprint(s)

(c) Agile software development has come to value:

- Individuals and interactions over processes and tools
- Working software over comprehensive documentation
- Customer collaboration over contract negotiation
- Responding to change over following a plan

Explain what each of these values mean.

THE FOUR VALUES OF THE AGILE MANIFESTO

The Agile Manifesto is comprised of four foundational values and 12 supporting principles which lead the Agile approach to software development. Each Agile methodology applies the four values in different ways, but all of them rely on them to guide the development and delivery of high-quality, working software.

1. Individuals and Interactions Over Processes and Tools

The first value in the Agile Manifesto is “Individuals and interactions over processes

and tools.” Valuing people more highly than processes or tools is easy to understand because it is the people who respond to business needs and drive the development process. If the process or the tools drive development, the team is less responsive to change and less likely to meet customer needs. Communication is an example of the difference between valuing individuals versus process. In the case of individuals, communication is fluid and happens when a need arises. In the case of process, communication is scheduled and requires specific content.

2. Working Software Over Comprehensive Documentation

Historically, enormous amounts of time were spent on documenting the product for development and ultimate delivery. Technical specifications, technical requirements, technical prospectus, interface design documents, test plans, documentation plans, and approvals required for each. The list was extensive and was a cause for the long delays in development. Agile does not eliminate documentation, but it streamlines it in a form that gives the developer what is needed to do the work without getting bogged down in minutiae. Agile documents requirements as user stories, which are sufficient for a software developer to begin the task of building a new function.

The Agile Manifesto values documentation, but it values working software more.

3. Customer Collaboration Over Contract Negotiation

Negotiation is the period when the customer and the product manager work out the details of a delivery, with points along the way where the details may be renegotiated. Collaboration is a different creature entirely. With development models such as Waterfall, customers negotiate the requirements for the product, often in great detail, prior to any work starting. This meant the customer was involved in the process of development before development began and after it was completed, but not during the process. The Agile Manifesto describes a customer who is engaged and collaborates throughout the development process, making. This makes it far easier for development to meet their needs of the customer. Agile methods may include the customer at intervals for periodic demos, but a project could just as easily have an end-user as a daily part of the team and attending all meetings, ensuring the product meets the business needs of the customer.

4. Responding to Change Over Following a Plan

Traditional software development regarded change as an expense, so it was to be avoided. The intention was to develop detailed, elaborate plans, with a defined set of features and with everything, generally, having as high a priority as everything else, and with a large number of many dependencies on delivering in a certain order so that the team can work on the next piece of the puzzle.

With Agile, the shortness of an iteration means priorities can be shifted from iteration to iteration and new features can be added into the next iteration. Agile's view is that changes always improve a project; changes provide additional value.

Perhaps nothing illustrates Agile's positive approach to change better than the concept of Method Tailoring, defined in An Agile Information Systems Development Method in use as: "A process or capability in which human agents determine a system development approach for a specific project situation through responsive changes in, and dynamic interplays between contexts, intentions, and method fragments." Agile methodologies allow the Agile team to modify the process and make it fit the team rather than the other way around.

THE TWELVE AGILE MANIFESTO PRINCIPLES

The Twelve Principles are the guiding principles for the methodologies that are included under the title "The Agile Movement." They describe a culture in which change is welcome, and the customer is the focus of the work. They also demonstrate the movement's intent as described by Alistair Cockburn, one of the signatories to the Agile Manifesto, which is to bring development into alignment with business needs.

The twelve principles of agile development include:

1. **Customer satisfaction through early and continuous software delivery** – Customers are happier when they receive working software at regular intervals, rather than waiting extended periods of time between releases.
2. **Accommodate changing requirements throughout the development process** – The ability to avoid delays when a requirement or feature request changes.
3. **Frequent delivery of working software** – Scrum accommodates this principle since the team operates in software sprints or iterations that ensure regular delivery of working software.
4. **Collaboration between the business stakeholders and developers throughout the project** – Better decisions are made when the business and technical team are aligned.
5. **Support, trust, and motivate the people involved** – Motivated teams are more likely to deliver their best work than unhappy teams.
6. **Enable face-to-face interactions** – Communication is more successful when development teams are co-located.
7. **Working software is the primary measure of progress** – Delivering functional software to the customer is the ultimate factor that measures progress.

8. **Agile processes to support a consistent development pace** – Teams establish a repeatable and maintainable speed at which they can deliver working software, and they repeat it with each release.
9. **Attention to technical detail and design enhances agility** – The right skills and good design ensures the team can maintain the pace, constantly improve the product, and sustain change.
10. **Simplicity** – Develop just enough to get the job done for right now.
11. **Self-organizing teams encourage great architectures, requirements, and designs** – Skilled and motivated team members who have decision-making power, take ownership, communicate regularly with other team members, and share ideas that deliver quality products.
12. **Regular reflections on how to become more effective** – Self-improvement, process improvement, advancing skills, and techniques help team members work more efficiently.

The intention of Agile is to align development with business needs, and the success of Agile is apparent. Agile projects are customer focused and encourage customer guidance and participation. As a result, Agile has grown to be an overarching view of software development throughout the software industry and an industry all by itself.

Q.3. (a) For what types of system are agile approaches to development particularly likely to be successful?

the most appropriate projects for agile are ones with aggressive deadlines, a high degree of complexity, and a high degree of novelty (uniqueness) to them

Examples of projects where Agile is suitable or may be possible:

1. Small to medium-sized software developments.
2. Product development where multiple variants are required or desirable.
3. Where the main deliverable can be broken down and produced (and even potentially released or at least accepted by the end customer) in incremental discrete packages. There is one major consideration here though; the requirements and functions developed during each iteration must be stand-alone and not have major dependencies with other requirements and functions outside of the current iteration. At the end of the iteration, we should have elements of product that are ready to release and deploy. If we cannot achieve that, we may not be using Agile at all.

• Small and medium-sized software product development.

• Custom software development in an organization where there is a clear commitment from customers to become involved in the development process.

https://quizlet.com/165864917/software-engineering-chapters-3-4-flash-cards/

(b) List and justify 4 questions that should be asked when deciding whether or not to adopt an agile method of software development.

Any 4 from those below. Others are also possible

- Is an incremental delivery strategy realistic?
- What type of system is being developed?
- What is the expected system lifetime?
- How is the development team organized?
- Is the system subject to external regulation?
- How large is the system that is being developed?

Q #1) Do you have the patronage from Top management?

As we can see, Agile is 80% delivery while only 20% quota of efforts is remaining for assessment, planning, and deployment. So, it is mandatory to have your top people, authorized and responsible for delivery, on board.

They should not only be in agreement with Agile delivery rather should be excited enough to sponsor the transition.

If they are not committed towards Agile and endorsing it just for the experimentation purpose, they would most probably opt for traditional processes after few failures rather than converting them into success through retrospectives.

Q #2) Can you switch to Task culture?

Organizational culture is an under-discussed but highly relevant factor in adopting agile according to my experience. If you are not willing or capable enough to modify your power structure, then Agile adoption is barely possible in its true spirit.

The most widely used classification of organizational culture is mentioned below.

There are four types of organizational cultures varying mainly in their distribution of powers within an organization. Most of the software companies, following conventional methodologies, have **"Role" culture** which is also termed as a Bureaucratic culture.

Q #3) What's the size and nature of your projects?

One of the myths about Agile adoption is that there is a Yes/No answer to this question. In reality, you can neither rule out Agile nor adopt it in its entirety. Rather, you have to decide the extent to which you can implement it for different projects and also the particular flavor of Agile.

After convincing top management to move towards Agile, I was pretty excited to embed it in all the projects right away. Then there was some divine assistance that asked me to experiment with a smaller startup project first. Once, we had a success story, we intended to replicate the same practices across all projects.

Q #4) Do your customers and suppliers (developers) buy-in the notion?

Since we have discussed the role of top management and middle management, it is the time to highlight the importance of two other major stakeholders' consent in moving to Agile i.e. customers and the software development, team.

Although customers are not directly executing any development process, they are not irrelevant in this decision at all. However, he needs to be in agreement with a couple of implications of using Agile. First, is not following the contract as Bible and the other is an Agile delivery approach.

Q #5) How big is your project team?

The most problematic thing about Agile is its compatibility with larger and distributed teams. Since the need for collaboration is so vital to the success of Agile, it becomes impossible at times to maintain it to the required level unless your project is intrinsically small enough to afford smaller team or can be broken down into autonomous smaller units.

There have been many surveys and studies on the relationship between the success of Agile and the teams' size or location. As you can see in figure 6, Agile delivers the most value when it is applied to co-located teams. But when the teams are geographically distributed, it has little or no edge over other processes.

(c) Identify and explain three important characteristics of extreme programming.

- Techniques and principles from Extreme Programming are now some of the most popular in Agile software development and focus on the following key themes:
  - Frequent releases, short development cycles
  - Pair programming
  - Regular builds and integration tests
  - Quality and avoiding code breakage
  - Simplicity of code, coding only what is needed
  - Rapid and regular feedback

(d) What are the barriers to introducing agile methods into large companies?

Large companies need to decide the specifications of the system prior to the system design. Any later change in system specification requires a new contract to be signed between company and contractor which is often painful affort for large companies. Therefore, large companies do not really prefer incremental delivery approach of agile development.

Q.4. (a) Explain how the principles underlying agile methods lead to the accelerated development and deployment of software.

a) *Individual and interactions over processes and tools*. By taking advantages of individual skills and ability and by ensuring that the development team knows what each other are doing, the overheads of formal communication and process assurance are avoided. This means that the team can focus on the development of working software.

b) *Working software over comprehensive documentation*. This contributes to accelerated development because time is not spent developing, checking and managing documentation. Rather, the programmer's time is focused on the development and testing of code.

c) *Customer collaboration over contract negotiation*. Rather than spending time developing, analyzing and negotiating requirements to be included in a system contract, agile developers argue that it is more effective to get feedback from customer's directly during the development about what is required. This allows useful functionality to be developed and delivered earlier than would be possible if contracts were required.

d) *Responding to change over following a plan*. Agile developers argue (rightly) that being responsive to change is more effective than following a plan-based process because change is inevitable whatever process is used. There is significant overhead in changing plans to accommodate change and the inflexibility of a plan means that work may be done that is later discarded.

(b) Extreme programming expresses user requirements as stories, with each story written on a card. Discuss the advantages and disadvantages of this approach to requirements description.

Advantages of stories:
1. They represent real situations that commonly arise so the system will support the most common user operations.
2. It is easy for users to understand and critique the stories.
3.They represent increments of functionality □ implementing a story delivers some value to the user.

Disadvantages of stories
1. They are liable to be incomplete and their informal nature makes this incompleteness difficult to detect.
2. They focus on functional requirements rather than non functional requirements.
3. Representing crosscutting system requirements such as performance and reliability is impossible when stories are used.
4. The relationship between the system architecture and the user stories is unclear so architectural design is difficult.

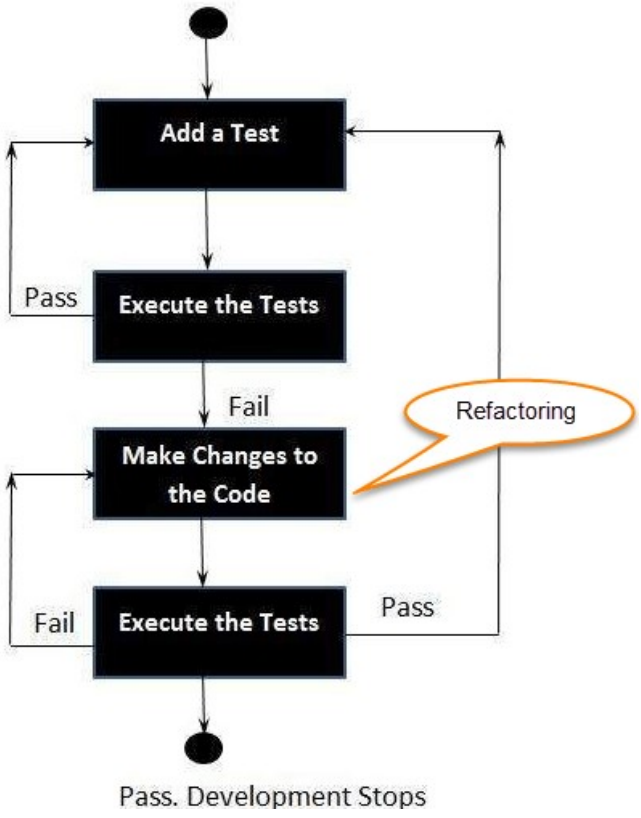
Q.5. (a) Identify and describe the main activities in Test Driven Development.

- Write User Stories
- Develop Product Skeletons
- Create Test Cases
- Design Test Cases
  - Test Design Template
  - Test Techniques
- Write JUnits
- Run Junit
- Write Product Code

FROM WEB

Following steps define how to perform TDD test,

1. Add a test.
2. Run all tests and see if any new test fails.
3. Write some code.
4. Run tests and Refactor code.
5. Repeat.



***Not the answer we are looking for but you should read it**

TDD Vs. Traditional Testing

TDD approach is primarily a specification technique. It ensures that your source code is thoroughly tested at confirmatory level.

- With traditional testing, a successful test finds one or more defects. It is same as TDD. When a test fails, you have made progress because you know that you need to resolve the problem.
- TDD ensures that your system actually meets requirements defined for it. It helps to build your confidence about your system.
- In TDD more focus is on production code that verifies whether testing will work properly. In traditional testing, more focus is on test case design. Whether the test will show the proper/improper execution of the application in order to fulfill requirements.
- In TDD, you achieve 100% coverage test. Every single line of code is tested, unlike traditional testing.
- The combination of both traditional testing and TDD leads to the importance of testing the system rather than perfection of the system.
- In Agile Modeling (AM), you should "test with a purpose". You should know why you are testing something and what level its need to be tested.

(b) Identify and describe the steps within the INVEST acronym that can be employed to measure User Story effectiveness.

1. **Independent**—The user story must be able to stand alone. It must be a feature or a component of a feature that can be tested and implemented as a unique element. To the extent possible, user stories should not be dependent on other activities. Ideally, they are written so that they can be delivered in any order.
2. **Negotiable**—As mentioned, a user story should invite collaboration and discussion about the best way to solve the business problem that is presented. The team, the Scrum master, and the product owner must be open to conversation about available options.

3. **Valuable**—The reason why we do anything in Agile is to drive business value, and the more business value being delivered, the higher the priority of the story. If the story does not add business value, the team should not work on it.
4. **Estimatable**—Is story too big or too vague? It must be clear enough that the developers and testers can reasonably estimate the complexity and length of time to deliver.

5. **Small**—The story should be small enough to be completed within a single sprint or iteration.
6. **Testable**—It is enough of a feature, and it is written in such a way that it can be tested to make sure it works as expected.

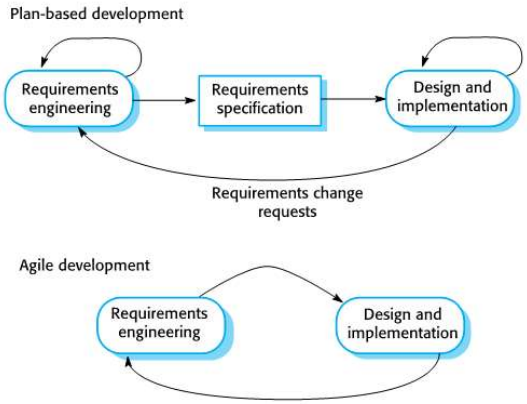
Q.6. (a) Create four child user stories from the following epic user story:

"As a business traveler, I want to be alerted if weather is likely to delay my flight."

(b) For each of child user stories that you just created write three to five acceptance test criteria for them.

Q.7.

(a) Briefly differentiate between Plan-Driven and Agile processes.



- Plan-driven processes are processes where all of the process activities are planned in advance and progress is measured against this plan.
- In agile processes, planning is incremental and it is easier to change the process to reflect changing customer requirements.
- In practice, most practical processes include elements of both plan-driven and agile approaches.
- There are no right or wrong software processes.
- Plan-driven development
  - A plan-driven approach to software engineering is based around separate development stages with the outputs to be produced at each of these stages planned in advance.
  - Not necessarily waterfall model – plan-driven, incremental development is possible
  - Iteration occurs within activities.
- Agile development
  - Specification, design, implementation and testing are inter-leaved and the outputs from the development process are decided through a process of negotiation during the software development process.

(b) Explain how the principles underlying agile methods lead to the accelerated development and deployment of software.

The principles underlying agile development are:

a) *Individual and interactions over processes and tools*. By taking advantages of individual skills and ability and by ensuring that the development team knows what each other are doing, the overheads of formal communication and process assurance are avoided. This means that the team can focus on the development of working software.

b) *Working software over comprehensive documentation*. This contributes to accelerated development because time is not spent developing, checking and managing documentation. Rather, the programmer's time is focused on the development and testing of code.

c) *Customer collaboration over contract negotiation*. Rather than spending time developing, analyzing and negotiating requirements to be included in a system contract, agile developers argue that it is more effective to get feedback from customer's directly during the development about what is required. This allows useful functionality to be developed and delivered earlier than would be possible if contracts were required.

d) *Responding to change over following a plan*. Agile developers argue (rightly) that being responsive to change is more effective than following a plan-based process because change is inevitable whatever process is used. There is significant overhead in changing plans to accommodate change and the inflexibility of a plan means that work may be done that is later discarded.

(c) Giving reasons for your answer based on the type of system being developed, suggest the most appropriate generic software process model that might be used as a basis for managing the development of the following systems:

- A system to control anti-lock braking in a car

• A virtual reality system to support software maintenance

• A university accounting system that replaces an existing system

• An interactive travel planning system that helps users plan journeys with the lowest environmental impact.

a) *Anti-lock braking system* This is a safety-critical system so requires a lot of up-front analysis before implementation. It certainly needs a plan-driven approach to development with the requirements carefully analysed. A waterfall model is therefore the most appropriate approach to use, perhaps with formal transformations between the different development stages.

b) *Virtual reality system* This is a system where the requirements will change and there will be an extensive user interface components. Incremental development with, perhaps, some UI prototyping is the most appropriate model. An agile process may be used.

c) *University accounting system* This is a system whose requirements are fairly well-known and which will be used in an environment in conjunction with lots of other systems such as a research grant management system. Therefore, a reuse-based approach is likely to be appropriate for this.

d) *Interactive travel planning system* System with a complex user interface but which must be stable and reliable. An incremental development approach is the most appropriate as the system requirements will change as real user experience with the system is gained.

Q.9.

(a) It has been suggested that one of the problems of having a user closely involved with a software development team is that they 'go native'. That is, they adopt the outlook of the development team and lose sight of the needs of their user colleagues. Suggest three ways how you might avoid this problem and discuss the advantages and disadvantages of each approach.

1. Involve multiple users in the development team. Advantages are you get multiple perspectives on the problem, better coverage of user tasks and hence requirements and less likelihood of having an atypical user. Disadvantages are cost, difficulties of getting user engagement and possible user conflicts.
2. Change the user who is involved with the team. Advantages are, again, multiple perspectives. Disadvantages are each user takes time to be productive and possible conflicting requirements from different users.
3. Validate user suggestions with other user representatives. Advantages are independent check on suggestions; disadvantage is that this slows down the development process as it takes time to do the checks.

(b) Extreme programming expresses user requirements as stories, with each story written on a card. Discuss the advantages and disadvantages of this approach to requirements description.

Advantages of stories:

1. They represent real situations that commonly arise so the system will support the most common user operations.
2. It is easy for users to understand and critique the stories.
3. They represent increments of functionality implementing a story delivers some value to the user.

Disadvantages of stories

1. They are liable to be incomplete and their informal nature makes this incompleteness difficult to detect.
2. They focus on functional requirements rather than non functional requirements.

3. Representing crosscutting system requirements such as performance and reliability is impossible when stories are used.
4. The relationship between the system architecture and the user stories is unclear so architectural design is difficult.

(c) Compare and contrast the V Model of Software Development to the Scrum Process. Support your answer by clearly identifying and describing the phases of both the V Model and Scrum process.

Q.10.

(a) What is "version control"?

Version control is a system that records changes to a file or set of files over time so that you can recall specific versions later.

- It allows you to
- revert files back to a previous state,
- revert the entire project back to a previous state,
- compare changes over time,
- see who last modified something that might be causing a problem,
- see who introduced an issue and when, and more.
- Using a VCS also generally means that if you screw things up or lose files, you can easily recover.
- Many people's version-control method of choice is to copy files into another directory (perhaps a time-stamped directory, if they're clever).
- This approach is very common because it is so simple, but it is also incredibly error prone.
- It is easy to forget which directory you're in and accidentally write to the wrong file or copy over files you don't mean to.

(b) Suggest four reasons why the productivity rate of programmers working as a pair might be more than half that of two programmers working individually.

Reasons why pair programming may be more efficient as the same number of programmers working individually:

1. Pair programming leads to continuous informal reviewing. This discovers bugs more quickly than individual testing.

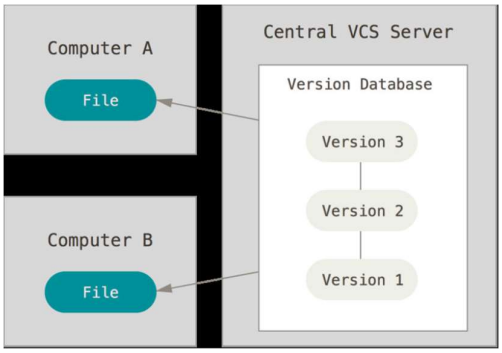
2. Information sharing in pair programming is implicit – it happens during the process. This reduces the need for documentation and the time required if one programmer has to pick up another's work. Individual programmers have to spend time explicitly sharing information and they are not being productive when doing so..

3. Pair programming encourages refactoring (the code must be understandable to another person). This reduces the costs of subsequent development and change and means that future changes can be made more quickly. Hence, efficiency is increased.

4. In pair programming, people are likely to spend less time in fine-grain optimization as this does not benefit the other programmer. This means that the pair focus on the essential features of the system which they can then produce more quickly.

(c) Compare and contrast Centralized Version Control Systems with Distributed Version Control Systems.

CENTRALIZED VERSION CONTROL

- The next major issue that people encounter is that they need to collaborate with developers on other systems.
- To deal with this problem, Centralized Version Control Systems (CVCSs) were developer
- These systems, such as CVS have a single server that contains all the versioned files, and a number of clients that check out files from that central place.
- For many years, this has been the standard for version control.
- 

- This setup offers many advantages, especially over local VCSs:
    - everyone knows to a certain degree what everyone else on the project is doing.

- administrators have fine-grained control over who can do what; and
- it's far easier to administer a CVCS than it is to deal with local databases on every client.
- However, this setup also has some serious downsides, the most obvious is the single point of failure that the centralized server represents:
- If that server goes down for an hour, then during that hour nobody can collaborate at all or save versioned changes to anything they're working on.
- If the hard disk the central database is on becomes corrupted, and proper backups haven't been kept, you lose absolutely everything – the entire history of the project except whatever single snapshots people happen to have on their local machines.

- Local VCS systems suffer from this same problem – whenever you have the entire history of the project in a single place, you risk losing everything.

Distributed Version Control Systems

- This is where Distributed Version Control Systems (DVCSs) step in.
- In a DVCS, such as Git,, clients don't just check out the latest snapshot of the files: they fully mirror the repository.
- Thus if any server dies, and these systems were collaborating via it, any of the client repositories can be copied back up to the server to restore it.
- Every clone is really a full backup of all the data.