2.3 Give three reasons why extreme programming, as envisaged by its developers, is not widely used

Extreme Programming (XP) is an agile framework that emphasizes both the broader philosophy of agile to produce higher-quality software to please customers and the more specific goal of making life better for the engineers developing it. The main characteristics of XP include dynamically changing software requirements; using a small, collocated extended development team; and leveraging technology that facilitates automated unit and functional tests. Reason for not widely used
De-emphasis on up-front design work: Developers sometimes take a myopic approach and fail to think ahead, but skipping analysis and design is a massive cop-out. When requirements change mid-project, developers often think it's because customers didn't know what they wanted.

Reliance on refactoring and unit testing to create a design: This is a mind-numbingly bad approach. Refactoring and unit testing have their places.

Dependence upon optional scope contracts to circumvent the problem of not knowing what will be delivered until the last minute: Programmers can use "optional scope contracts" to goof off for six months and turn in a high-quality login screen at the end.

Catching the inevitable bugs in a design created from refactoring with pair programming

XP is too minimal and too risky: XP is limited to chaotic projects that previously had no process. But then again, on such projects, anything would be an improvement.

XP is complex: The practices and values are simple, but it takes a lot of concentration and effort to steer a "true" XP project.

XP relies on too many things: XP is a minimalist process that relies on several practices to make up for its lack of rigor in other areas.

2.8Why are daily scrums likely to reduce the time that is normally required for new team members to become productive?

The daily Scrum is a time-boxed fifteen-minute stand-up meeting between the development team. It is meant for the team to check in and state what they accomplished the day before, what they will be working on today, and what blocks are impeding their progress.
Advantages of having scrum calls:

- It gives a snapshot view on the progress of the team and how they are doing in their sprint
- They allow the team to review and adapt the sprint backlog
- It gives insight into how quickly the team is moving through their assigned tickets

- It helps the team to coordinate the work being done and to plan their days
- Progress in the sprint is managed to make sure everyone is still on track with meeting the sprint goal
- It gives an early indicator, if the team will complete their sprint on time or if there will be unfinished tasks at the end of the sprint
- The team members can discuss any blockers that might be preventing them from completing their tasks that need to be dealt with

As you can see the daily Scrums are essential in keeping the team on track and making sure the sprints are moving forward smoothly. Therefore, they should not be skipped. Communication is a major factor in success and the daily Scrum helps keep the communication lines open within the team. Daily scrums provide the opportunity for the team members to think, reflect on, and adjust the work done on a project. This opportunity helps new team members to learn fast because of the interaction they share with experienced team members. This means that the experience of other team members robs the new team members during timeboxed scrums, making them to become more productive.

3.8 Explain why domain knowledge is important when identifying and designing product features.

A domain in this context refers to a business sector such as Manufacturing, Healthcare, Banking, Insurance, and so on and so forth. Domain knowledge points to the comprehension and understanding of the inner workings, processes, procedures, and other key aspects of an enterprise. For instance, professionals seeking domain knowledge in retail banking would need to understand the transactions concerned with retail banking web applications such as fund transfers and currency conversions.

To identify the product features and design, one should be **independent** where the implementation of a feature must not depend on the implementation of other features or the order in which those features are activated. **Coherent where the** Functionality should be tied to a single feature. It is crucial that they don't do more than one thing and that they don't have side effects. **Relevant** for features to be useful, they should reflect how users typically perform certain tasks. Functions that are rarely used shouldn't be offered**.**

3.9 Suggest how a development team might avoid feature creep when “it is” to be in agreement with “a team” faced with many different suggestions for new features to be added to a product.

Feature creep, or scope creep, refers to the excessive addition of new features when developing a product. Adding new features feels like a great way to improve your product, but this philosophy can backfire when those features grow unwieldy. Feature creep unnecessarily delays product launches, drives up costs, and may even destroy a project. As a developer or designer, it’s critical to avoid scope creep. To avoid feature creep, the product manager and the development team should review all feature proposals and compare new proposals and features that have already been accepted for implementation. we can also avoid this by

- Streamline your design process
- Focus on fundamentals to avoid feature creep
- Balance features and functions
- Implement a feature approval process
- Focus on priorities
- Listen to the feedback
- Examine the impact

By considering some questions

1. Does this feature really add anything new or is it simply an alternative way of doing something that is already supported?
2. Is this feature likely to be important to and used by most software users?
3. Can this feature be implemented by extending an existing feature rather than adding another feature to the system?
4. Does this feature provide general functionality or is it a very specific feature?

4.3 Why is it important to try to minimize complexity in a software system?

Software complexity is a natural byproduct of the functional complexity that the code is attempting to enable. With multiple system interfaces and complex requirements, the complexity of software systems sometimes grows beyond control, rendering applications and portfolios overly costly to maintain and risky to enhance. software complexity can run rampant in delivered projects, leaving behind bloated, cumbersome applications. it is important to minimize.

Complexity is related to the number of relationships between elements in a program and the type and nature of these relationships. Complexity is also affected by the type of relationship. The static relationship does not depend on program execution whereas a dynamic relationship which changes over time, is more complex than static. Complexity led to a programming error also leads to the high maintenance cost and it makes it difficult to enhance and test.