Name :Sardar Hassan aftab

Roll no : 14873

Subject : Software Engnerring

Section : D

Class : BS-CS “ 3”

Chapter : 1

Introduction

1.1.

Explain why professional software that is developed for a customer is not simply the programs that have been developed and delivered.

Solution :

Professional software developed for a customer is not merely the set of programs delivered because it encompasses a broader scope of responsibilities and deliverables essential to meeting the customer’s needs effectively and ensuring the software’s longevity and maintainability.

Here are key reasons why professional software involves more than just the code:

Aspect	Explanation
Requirements Analysis	Professional software starts with in-depth requirements gathering to understand customer needs, domain, and constraints, ensuring that the software aligns with their goals.
Documentation	Comprehensive documentation, including user manuals, design documents, and maintenance guides, is essential for users, developers, and future maintainers. This extends beyond the program code itself.
Testing and Quality Assurance	Rigorous testing is conducted to ensure reliability, performance, and security. This includes unit tests, integration tests, and user acceptance testing, ensuring the software performs as expected.
Maintenance and Support	Professional software development includes long-term maintenance and support plans. It requires bug fixes, updates for compatibility with new systems, and potential feature enhancements over time.
Deployment and Training	It involves deployment strategies and often includes training sessions to ensure the customer can use the software effectively.
Project Management	Managing timelines, budgets, resources, and communication with stakeholders is crucial to delivering a successful project, making it more than just a collection of code files.
Compliance and Security	Ensuring the software meets industry regulations, security standards, and compliance requirements is a vital part of professional software that protects both the developer and the customer.

Each of these aspects is integral to professional software and contributes to its quality, usability, and sustainability, distinguishing it from standalone programs.

1.2.

What is the most important difference between generic software product development and custom software development? What might this mean in practice for users of generic software products?

Answer :

The most important difference between generic software product development and custom software development is **the target audience and adaptability**:

Generic Software :

Generic Software is developed for a broad audience with a wide range of potential users. It is designed to meet the general needs of many users across different industries or use cases.

Custom Software :

Custom Software is developed specifically for a single customer or a specific organization, tailored precisely to their unique requirements.

In practice, this distinction has several implications for users of generic software products:

Aspect	Implications for Generic Software Users
Flexibility	- Users often need to adapt workflows to fit the software. - May lack specific features for unique processes.
Customization	- Limited options for personalizing features. - Changes must suit a broad audience, not individual needs.
Features	- Might lack niche or industry-specific functionalities. - Often includes unnecessary features for some users.
Updates	- Updates are based on general market demand. - User-specific needs may not be addressed promptly.
Support	- Support is broad and may not address unique issues. - Training is generic, leading to a steeper learning curve.

In practice, users may have to adjust to limitations or use workarounds for specific needs.

1.3

.Briefly discuss why it is usually cheaper in the long run to use software engineering methods and techniques for software systems.

Answer :

Using software engineering methods and techniques is generally cheaper in the long run because it ensures a structured, disciplined approach to development that minimizes costly issues over time. Here's why:

Aspect	Explanation
Reduced Maintenance Costs	Well-designed, documented, and tested software is easier and cheaper to maintain and update, reducing long-term costs.
Lower Error Rate	Following engineering principles (like testing and quality assurance) reduces bugs and system failures, avoiding expensive fixes after deployment.
Scalability	Software engineering methods promote modular, scalable designs, making it easier and less costly to add features as needs evolve.
Enhanced Productivity	Clear processes and standards improve team productivity, reducing development time and labor costs.

Better Risk Management	Identifying and mitigating risks early reduces the likelihood of unexpected expenses and project delays.

Overall, investing in software engineering methods leads to reliable, maintainable systems, minimizing unforeseen expenses and making it more cost-effective in the long term.

1.4.

Software engineering is not only concerned with issues like system heterogeneity, business and social change, trust, and security, but also with ethical issues affecting the domain. Give some examples of ethical issues that have an impact on the software engineering domain.

Answer :

Ethical issues in software engineering can have significant consequences for society, users, and organizations. Here are some key ethical concerns that impact the domain:

Ethical Issue	Explanation
Privacy and Data Protection	Ensuring user data is collected, stored, and used responsibly and with consent to avoid misuse, breaches, or surveillance.
Bias and Fairness	Avoiding biased algorithms or datasets that can lead to unfair treatment of certain groups in applications like hiring or loan approval.
Transparency and Accountability	Making software decisions (like AI-based outcomes) understandable and accountable to end-users and stakeholders.
Security and Safety	Implementing rigorous security measures to protect against breaches, which could cause harm to individuals or critical systems.
Intellectual Property and Plagiarism	Respecting ownership of code, avoiding plagiarism, and adhering to licensing agreements for software and open-source components.
Environmental Impact	Designing software to minimize resource usage (e.g., energy consumption), reducing its environmental footprint.

These ethical issues require careful consideration to build software that is not only functional but also responsible and aligned with societal values.

1.5.

Based on your own knowledge of some of the application types discussed in Section 1.1.2, explain, with examples, why different application types require specialized software engineering techniques to support their design and development.

Answer :

Different application types require specialized software engineering techniques to support their design and development. Examples include mobile applications, web applications, and enterprise applications.

Explanation:

Different application types require specialized software engineering techniques to support their design and development due to their unique needs and requirements. Here are some examples:

1.Mobile applications:

Mobile apps are designed for specific platforms such as iOS or Android. They require specialized techniques to optimize performance, user experience, and adapt to different screen sizes.

2. Web applications :

Web apps are accessed through web browsers and often involve complex interactions and data processing. They require specialized techniques for front-end development, back-end integration, security, and scalability.

3. Enterprise applications:

Enterprise software is designed to manage complex business processes and integrate with other systems. It requires specialized techniques for customization, integration, security, and high availability.

1.6.

Explain why the fundamental software engineering principles of process, dependability, requirements management, and reuse are relevant to all types of software system.

Answer :

The fundamental software engineering principles of **process**, **dependability**, **requirements management**, and **reuse** are relevant to all types of software systems because they ensure consistency, reliability, alignment with user needs, and efficiency. Here's why each principle is universally applicable:

Principle	Relevance Across Software Types
Process	A defined development process provides structure, guiding teams through planning, development, testing, and deployment stages. It reduces project risk, improves quality, and ensures timely delivery, regardless of application type.
Dependability	All software must be reliable, safe, and secure, as failures can lead to user dissatisfaction, financial loss, or even harm in critical systems. Dependability ensures software remains functional, secure, and resilient in all scenarios.
Requirements Management	Capturing, tracking, and managing requirements is essential for delivering software that meets user needs. It prevents misalignment and scope creep, ensuring that the final product satisfies customer expectations and operational goals.
Reuse	Reusing components, code, or frameworks saves development time, reduces cost, and enhances consistency across projects. It also allows leveraging proven, reliable code, which improves overall software quality and maintainability.

1.7.

Explain how electronic connectivity between various development teams can support software engineering activities.

Answer :

Electronic connectivity between various development teams greatly enhances software engineering activities by enabling better collaboration, communication, and access to resources. Here's how:

2. **Improved Communication**:

Teams can communicate in real-time using platforms like chat, video calls, or emails, facilitating quick issue resolution, sharing of updates, and alignment of efforts across locations. This minimizes delays and misunderstandings that can occur in traditional face-to-face communication.

3. **Collaboration and Sharing**:

Tools such as version control systems (e.g., Git) and project management platforms (e.g., Jira, Trello) enable teams to collaborate

seamlessly. These platforms allow for task management, progress tracking, and sharing of resources like code or documentation, making it easier for teams to stay aligned and work together efficiently.

4. **Access to Centralized Resources**:

Cloud-based repositories and shared documentation systems ensure that all teams have access to the most current resources, such as codebases, design documents, and specifications. This centralized access reduces the chances of miscommunication or working with outdated information.

5. **Time Zone Flexibility**:

Teams spread across different time zones can contribute to a project without delays, using a "follow-the-sun" approach. This ensures that work continues around the clock, speeding up the development cycle and reducing project timelines.

6. **Faster Problem Resolution**:

With teams collaborating remotely, issues can be addressed faster by drawing on a wider pool of expertise. Developers, designers, and other specialists can provide input and solutions in real-time, which accelerates problem-solving.

7. **Better Quality Control**:

Continuous integration and automated testing tools, which can be accessed by remote teams, help in detecting and resolving issues early in the development process. This ensures that the software is more stable and of higher quality before reaching production.

Overall, electronic connectivity supports a more agile, coordinated, and efficient software development process, allowing teams from different locations to contribute effectively, speed up workflows, and maintain high standards.

1.8.

Noncertified individuals are still allowed to practice software engineering. Discuss some of the possible drawbacks of this.

Answer :

Allowing noncertified individuals to practice software engineering can lead to several potential drawbacks. While certification is not a strict requirement in many areas, the absence of certification may impact the quality and reliability of software development in the following ways:

1. **Lack of Standardized Knowledge**:

   Certified professionals typically undergo rigorous training that covers industry best practices, methodologies, and technical standards. Without certification, noncertified individuals may lack a comprehensive understanding of key concepts, leading to inconsistent practices, inefficient development processes, and suboptimal design decisions.

2.Increased Risk of Errors:

Software engineering requires attention to detail, thorough testing, and error-free implementation. Noncertified individuals might not be as familiar with established practices like testing protocols, version control, or code review techniques, which can result in higher error rates, bugs, or system failures that are difficult to debug or fix.

3.Security Vulnerabilities:

Proper understanding of security best practices is critical in software development. Noncertified developers may not be fully aware of security risks, such as data breaches or vulnerabilities in the code, leading to insecure software that is vulnerable to exploitation by attackers.

4.Reduced Accountability:

Certified professionals are typically held to a higher standard of accountability, often due to their formal education or professional licensing. Noncertified developers might not be subject to the same level of scrutiny, making it harder to ensure consistent quality control or accountability when issues arise in software systems.

5.Missed Opportunities for Professional Growth:

Certification programs often provide developers with opportunities to learn the latest tools, techniques, and technologies. Noncertified developers might miss out on these educational opportunities, leading to stagnation in their skill development and potentially falling behind in the fast-evolving field of software engineering.

6 . Lack of Trust from Clients or Employers:

Many organizations and clients may prefer to work with certified professionals to ensure that software is being developed to the highest standards. Noncertified individuals may face challenges in gaining trust, especially for critical projects where quality and reliability are paramount.

7.Difficulty in Collaboration:

Working in teams often requires adhering to specific standards and processes. Noncertified individuals may not be familiar with these established methodologies, creating inconsistencies in the way the team operates. This can slow down development, reduce communication effectiveness, and affect overall project outcomes.

While noncertified individuals can still contribute to software engineering, these potential drawbacks highlight the importance of industry-recognized certification in maintaining high standards of quality, security, and professionalism in software development.

1.9.

For each of the clauses in the ACM/IEEE Code of Ethics shown in Figure 1.4, propose an appropriate example that illustrates that clause.

Answer :

Public :

An example of acting in the public's interest is not to share any private information that the software engineer is privy to while working on a project.

Client and Employer : A good software engineer will not let the needs of the client or the wants of the employer harm the public.

Product :

A good software engineer would not allow software to be released that they know is faulty and will crash.

Judgement :

An example of using good judgement is, if a software engineer is approached by a client that wants them to build software that causes airplanes to crash, the software engineer knows that they must decline since although that may be in the client's interest, it is not in the public's.

Management :

For example, Software engineering managers and leaders will not encourage employees to slack off when they are aware there is a deadline coming up.

Profession :

For example, software engineers will not accept payment for inadequate software that they have admitted will not work as intended. This would reflect badly on other software engineers.

Colleagues :

An example of following this clause would be a software engineer not stealing their coworkers code without permission.

Self :

For example, a good software engineer will keep up with changes in technology and encourage others to do the same.

The development and deployment of drones, which are equipped with various software systems for tasks like navigation, sensing, and decision-making, present a range of societal challenges. These challenges extend beyond technical concerns and touch on ethical, legal, and social implications. Here are some of the key challenges:

1. Privacy Concerns :

Drones are often equipped with cameras, sensors, and other monitoring equipment that can potentially infringe on individuals' privacy. Drones flying over private property or public spaces could capture images, videos, or data without consent, leading to concerns about surveillance and data misuse.

Example: Drones used by businesses for surveillance or by governments for monitoring public spaces could collect personal data on unsuspecting individuals, raising questions about who owns the data and how it is used.

2. Safety and Security Risks

Drones can pose safety risks to both people and property. Malfunctions, crashes, or even malicious hacking could lead to accidents or intentional harm. In addition, drones equipped with surveillance tools could be hijacked and used for unauthorized surveillance or cyber-attacks.

- **Example**: A drone malfunctioning in a crowded area could cause physical harm to people, or a drone used for delivery could crash into buildings or vehicles. Additionally, drones

with access to sensitive data might become targets for cybercriminals.

3. Airspace Regulation and Control

The integration of drones into existing air traffic systems is a complex task. Drones need to operate safely alongside manned aircraft, and this requires proper regulation and coordination. There is a need for frameworks that define airspace boundaries, especially in urban environments, to avoid collisions and ensure safe operation.

- **Example**: The increasing use of drones for deliveries or recreational use could lead to overcrowded skies, making it harder to maintain safe distances between drones and manned aircraft, particularly in busy urban airspaces.

4. **Ethical and Legal Issue**:

The deployment of drones for military or law enforcement purposes raises ethical concerns about the use of force, surveillance, and accountability. Drones equipped with weapons, for example, could potentially be used in targeted killings, leading to debates about the ethics of remote warfare and the legal implications.

- **Example**: The use of drones in military operations or for law enforcement surveillance raises questions about accountability, particularly when drones are used for decisions that impact human lives, such as surveillance or targeting people.

5. Job Displacement :

As drones are increasingly used in various industries (like delivery, agriculture, and surveillance), they could lead to job displacement in sectors that rely on human workers. For instance, delivery drivers, aerial surveyors, or even pilots may face job losses as drones take over tasks previously performed by humans.

- **Example**: Drones replacing delivery trucks could lead to reduced demand for delivery drivers, impacting employment in logistics and transportation sectors.

6. Environmental Impact :

While drones are often seen as more environmentally friendly alternatives to other modes of transportation (like delivery trucks or planes), their widespread use may still pose environmental challenges. For example, the energy consumption of drones, especially those used for commercial purposes, could have an impact on energy resources and carbon footprints.

- **Example**: The production and operation of drones may involve the use of non-renewable resources, and the potential increase in their use could lead to higher electricity consumption or pollution, particularly in large-scale commercial applications.

7. Lack of Public Trust and Acceptance

Public concerns about privacy, safety, and the ethical use of drones can lead to a lack of trust in drone technologies. This could slow down the adoption of drones in various sectors, as individuals and communities may be hesitant to embrace technologies they perceive as invasive or unsafe.

- **Example**: Public opposition to drones flying in residential areas or being used for surveillance may require more robust regulations, transparency, and efforts to build trust among citizens.

8. Technical Challenges and Reliability :

Building reliable, autonomous software systems that control drones is complex. Drones must be able to navigate in changing environments, avoid obstacles, and make decisions based on real-time data. Failures in these systems could result in accidents or mission failures, which may harm people, damage property, or result in loss of data.

- **Example**: If a drone malfunctioned and crashed due to software or hardware failure during a delivery or surveillance mission, it could result in significant property damage, injury, or loss of valuable goods.

9. Security and Hacking Risks :

Drones rely on software and communication networks, making them vulnerable to hacking, unauthorized control, or spoofing of signals. This could allow malicious actors to hijack drones for criminal purposes, such as spying, sabotaging infrastructure, or smuggling contraband.

- **Example**: A drone used for delivery could be hijacked and diverted to a location by hackers, leading to theft, privacy violations, or even the use of the drone as a vehicle for smuggling.

Conclusion:

The societal challenges of building drone systems are multi-faceted and involve technical, ethical, regulatory, and social considerations. These issues require careful attention to ensure that the benefits of drones—such as increased efficiency, accessibility, and innovation—are realized while minimizing negative impacts on privacy, safety, and security. Addressing these challenges effectively will require collaboration among governments, industry, developers, and the public.

Chapter : 2

Exercises :

2.1. Suggest the most appropriate generic software process model that might be used as a basis
for managing the development of the following systems. Explain your answer according to the
type of system being developed:
A system to control antilock 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

Solution :

Here are the recommended software process models for each type of system, along with explanations based on the characteristics of each system:

System	Recommended Software Process Model	Explanation
Antilock braking control system	**V-Model**	This is a safety-critical, embedded system that requires thorough verification and validation. The V-Model is suitable due to its focus on rigorous testing at each development stage, ensuring reliability and adherence to safety standards.
Virtual reality system for software maintenance	**Prototyping Model**	VR systems require high levels of user interaction, visualization, and often experimental interfaces. The Prototyping Model allows for the iterative refinement of system requirements and design, accommodating feedback on usability and functionality throughout development.
University accounting system replacement	**Incremental Model**	Replacing an existing system often involves transferring established functionalities to a new system while minimizing disruptions. The Incremental Model allows for gradual migration and integration of features, making it easier to transition smoothly without halting operations.
Interactive travel planning system for environmental impact	**Agile Model**	This interactive, user-centered system may require frequent updates based on evolving user needs and environmental data. Agile's flexibility and iterative cycles allow for

		continuous feedback, adaptive planning, and quick response to changing requirements and data sources.

Each model is chosen to balance the needs for reliability, flexibility, and user-centered design specific to each system's requirements and context.

2.2. Incremental software development could be very effectively used for customers who do not
have a clear idea about the systems needed for their operations. Discuss

Solution :

Incremental software development is an effective approach for customers who lack a clear understanding of their system requirements due to its flexible, iterative nature, which offers several benefits in such cases:

1. Gradual Requirements Discovery :

Incremental development enables customers to clarify their needs by viewing tangible outputs in stages. Each increment provides a working part of the system, allowing the customer to refine requirements based on real functionality.

2. Reduced Risk of Misalignment :

Unclear requirements often lead to misaligned features. Incremental development mitigates this by focusing on small, manageable portions of the system, reducing misunderstandings and ensuring alignment with evolving needs.

3. Early Delivery of Key Features :

This approach allows delivery of core features early, giving the customer immediate value and helping them recognize additional requirements as they interact with the system.

4. Enhanced Flexibility and Responsiveness :

Iterative cycles make incremental development adaptable, allowing adjustments based on the customer’s feedback. This flexibility is essential for customers with evolving or unclear requirements.

5. Improved Stakeholder Engagement and Satisfaction :

Frequent customer involvement builds trust and increases satisfaction, as the system is shaped according to their evolving priorities.

In summary, incremental development offers a structured yet adaptable approach, ensuring the final system aligns with the customer’s needs as they become clearer.

2.3. Consider the integration and configuration process model shown in Figure 2.3. Explain why it is essential to repeat the requirements engineering activity in the process.

Solution :

In the integration and configuration process model, repeating the requirements engineering activity is essential because it ensures that the system aligns with changing requirements and constraints throughout the integration of off-the-shelf (OTS) components and configurable systems. Here's why this repetition is crucial:

1. Alignment with Evolving Customer Needs

- Customer needs often evolve as they see early versions or prototypes. Revisiting requirements ensures that the system adapts to these changing expectations, which is especially important when using OTS components with limited customization.

2. Addressing Compatibility and Integration Issues

- Each new OTS component or module introduced into the system may have unique requirements or limitations. By revisiting requirements engineering, the team can assess and adjust requirements to ensure compatibility across all components.

3. Managing Constraints of Configurable Components

- When working with configurable components, certain features may be limited or behave differently than initially expected. Re-evaluating requirements allows the team to accommodate these constraints or find alternative configurations to meet key requirements.

4. Ensuring Regulatory and Compliance Adherence

- Systems that must meet specific standards (e.g., security, performance, legal) may face changing regulatory requirements during development. Repeating requirements engineering helps the team stay compliant by re-aligning system requirements to new regulations as they arise.

5. Optimizing Cost and Performance

- Requirements engineering repetition provides an opportunity to assess cost and performance trade-offs based on available OTS components. This ensures the system design remains cost-effective without compromising essential functionality or quality.

In summary, repeating the requirements engineering activity in the integration and configuration model is essential to adapt to evolving needs, ensure

component compatibility, manage constraints, maintain compliance, and optimize the system's cost and performance as it integrates new elements.

2.4. Suggest why it is important to make a distinction between developing the user requirements and developing system requirements in the requirements engineering process.

Solution :

Distinguishing between user requirements and system requirements in the requirements engineering process is important because they serve different purposes and stakeholders, impacting the clarity, feasibility, and effectiveness of the development process. Here's why the distinction is essential:

1. Different Stakeholders and Perspectives

- **User Requirements** capture what the end-users need from the system to achieve their goals. These are high-level, often non-technical descriptions focused on usability and functional expectations.
- **System Requirements** are technical specifications derived from user requirements, detailing how the system should operate internally to meet user needs. System requirements guide developers, focusing on system architecture, performance, and integration.

2. Improved Clarity and Communication

- Separating user and system requirements avoids confusion by providing a clear understanding of what users want versus how the system will fulfill those needs. This division improves communication across technical and non-technical stakeholders, ensuring that user goals and technical details are well understood.

3. Enhanced Feasibility and Validation

- User requirements offer a basis for validating that the system will meet end-user needs, while system requirements enable feasibility analysis by examining how these needs can be technically implemented. This layered approach helps identify potential technical constraints early and allows adjustments to maintain user satisfaction within technical limitations.

4. Better Traceability and Requirement Management

- Maintaining separate user and system requirements improves traceability, making it easier to track changes from user expectations to technical implementations. This is especially valuable in complex projects, ensuring each user requirement is accurately represented in the system requirements and simplifying change management.

5. Improved Design Flexibility

- By defining user requirements first, system architects can explore multiple design options that meet the user's needs without being bound to a specific technical solution initially. This allows more flexibility in choosing optimal design approaches before finalizing technical details.

In summary, distinguishing user requirements from system requirements ensures clear communication, feasibility analysis, traceability, and design flexibility. It helps bridge the gap between user expectations and the technical specifications necessary to build a successful system.

2.5. Using an example, explain why the design activities of architectural design, database design, interface design, and component design are interdependent.

Solution :

The design activities of architectural, database, interface, and component design are interdependent because changes in one area affect the others. Here's a shorter example of how this works, using an e-commerce application:

1. **Architectural Design**
   The choice of architecture (e.g., microservices) divides the application into distinct services like product catalog and order processing, impacting database separation, interface endpoints, and component structure.
2. **Database Design**
   Database structure (e.g., tables for products and orders) influences how data is presented in the interface and how components access data, while adhering to architectural constraints.
3. **Interface Design**
   Interface requirements (e.g., product filters) determine the data needed from the database and the behavior of components, aligning with architecture choices for compatible UI options.
4. **Component Design**
   Each component (e.g., adding items to cart) depends on database structure, interacts with the user interface, and follows architecture protocols.

Summary Table

Design Activity	**Interdependencies Impact Example**
Architectural Design	Influences database structure, interface endpoints, and component scope.
Database Design	Affects interface data needs and component access patterns.
Interface Design	Drives database queries and component functionality requirements.
Component Design	Relies on database, interfaces with UI, and follows architectural protocols.

In short, changes in one design area require adjustments in others to maintain consistency and functionality.

2.6. Explain why software testing should always be an incremental, staged activity. Are programmers the best people to test the programs that they have developed?

Solution :

Why Software Testing Should Be Incremental and Staged

Software testing should be incremental and staged because it allows issues to be identified and resolved early in development, reducing the time and cost of fixing bugs. Each stage targets different aspects of the software, gradually building confidence in its functionality, performance, and reliability. For example:

- **Unit Testing:**

  Tests individual components or functions in isolation to catch small errors early.

- **Integration Testing:**

  Ensures that combined components work together, catching issues that arise from component interactions.

- **System Testing**:

  Tests the software as a whole to verify it meets requirements.

- **Acceptance Testing:**

  Confirms the software meets user needs and business requirements before deployment.

By staging tests, developers can detect issues at each layer of functionality, reducing the risk of complex, compounded bugs surfacing late in development.

Are Programmers the Best People to Test Their Own Code?

Generally, programmers are not the best people to test the programs they've developed because:

1. **Bias:**

   Programmers may unconsciously overlook errors in their own work due to familiarity or assumptions about how the code should function.

2. **Tunnel Vision:**

   They may focus on verifying that the software works as expected rather than probing for unexpected issues.

3. **Limited User Perspective:**

   Programmers may lack an end-user's perspective, so they may miss usability issues or scenarios that real users might encounter.

For these reasons, independent testers, who bring fresh eyes and a user-focused mind

2.7. Imagine that a government wants a software program that helps to keep track of the utilization of the country's vast mineral resources. Although the requirements put forward by the government were not very clear, a software company was tasked with the development of a prototype. The government found the prototype impressive, and asked it be extended to be the actual system that would be used. Discuss the pros and cons of taking this approach.

Solution :

Pros and Cons of Extending a Prototype to an Operational System

Pros:

1. **Faster Development**:

   Since the prototype is already built, extending it can save time compared to starting from scratch. This accelerated timeline allows the government to start tracking resource utilization sooner.

2. **Immediate Feedback**:

   The government's initial approval suggests the prototype meets some key requirements. This feedback helps refine the system in alignment with the government's expectations, improving the chances of delivering a suitable product.

3. **Cost-Effective**:

   By building upon the prototype, the development team can use existing code and design elements, reducing development costs as well as rework efforts.

4. **Clearer Requirements**:

   The prototype provides a tangible foundation that can help clarify requirements. Observing the prototype in action can highlight missing features or necessary changes that might otherwise go unnoticed.

Cons:

1. **Poorly Structured Code**:

   Prototypes are often developed quickly without robust architecture or scalability in mind. Extending a prototype to a full system might require significant restructuring, which could be time-consuming and costly.

2. **Lack of Documentation**:

   Prototypes often lack thorough documentation, which can make maintaining and extending the system challenging, especially for a complex, long-term government project.

3. **Hidden Technical Debt**:

   Prototypes may contain shortcuts or quick fixes that could become problematic as the system grows. Extending a prototype without addressing this technical debt could lead to performance issues, higher maintenance costs, and instability.

4. **Unclear Requirements**:

   Since the original requirements were vague, the prototype may only capture a portion of the needed functionality. Relying on it could lead to a system that fails to meet the government's evolving or more detailed needs.

Conclusion

While extending a prototype offers advantages in terms of speed and cost, the lack of scalability, documentation, and potential technical debt must be carefully considered. To mitigate risks, the software company might choose a hybrid approach: refactor critical parts of the prototype to ensure scalability, while using its core elements as a framework to clarify and expand upon the government's requirements.

2.8.

You have developed a prototype of a software system and your manager is very impressed by it. She proposes that it should be put into use as a production system, with new features added as required. This avoids the expense of system development and makes the system immediately useful. Write a short report for your manager explaining why prototype systems should not normally be used as production systems.

Solution :

Report: Why Prototypes Should Not Be Used as Production Systems

To: [Manager's Name]
From: [Your Name]
Date: [Insert Date]
Subject: Concerns About Using Prototype as a Production System

---

Introduction
While the prototype of the software system demonstrates promise, it is not suitable for immediate deployment as a production system. This report explains why prototype systems should not typically be used in a production environment without significant modifications.

---

1. Lack of Scalability
Prototypes are built to quickly demonstrate core functionality and may not be designed to handle large user loads, data volumes, or long-term growth. A production system must be scalable to meet the increasing demands of real-world use.

2. Insufficient Testing
Prototypes are often tested only for specific features and may not undergo thorough testing for reliability, security, and performance. A production system requires extensive testing to ensure stability under

diverse conditions and to avoid potential failures or security vulnerabilities.

3. Limited Error Handling and Robustness
Prototypes often lack comprehensive error handling and fault tolerance. A production system must be robust enough to gracefully handle errors, recover from failures, and maintain system integrity.

4. Poor Documentation and Maintainability
Prototypes are typically developed quickly and may not include necessary documentation or maintainable code. A production system requires clear documentation and well-structured code to facilitate future updates and troubleshooting.

5. Security Risks
Prototypes generally prioritize functionality over security. A production system must include robust security measures to protect sensitive data and prevent unauthorized access or attacks.

6. Incomplete Features
The prototype may not capture all the functional requirements of the final system. Directly deploying it could lead to missing features or inadequate support for business needs, requiring expensive rework post-deployment.

Conclusion
Although the prototype has demonstrated potential, using it as a production system without significant improvements in scalability, security, and reliability could lead to operational risks and inefficiencies. I recommend further development and refinement to ensure the system meets the necessary standards for production use.

Recommendations

- Refactor the prototype for scalability and robustness.
- Conduct extensive testing, including performance and security assessments.
- Add documentation and improve code maintainability.
- Clarify and incorporate additional business requirements.

I look forward to discussing how we can transition the prototype into a fully-fledged production system.

[Your Name]
[Your Position]

2.9. Suggest two advantages and two disadvantages of the approach to process assessment and improvement that is embodied in the SEI's Capability Maturity framework.

Solution :

The Capability Maturity Model (CMM) developed by the Software Engineering Institute (SEI) is a process assessment and improvement approach used to evaluate and enhance the maturity of an organization's processes. Here are two advantages and two disadvantages of the CMM approach:

Advantages:

Advantage	Description
Structured Process Improvement	CMM provides a clear, step-by-step framework for organizations to improve their processes, ensuring that improvements are systematic and measurable.
Improved Predictability and Quality	By focusing on improving process maturity, organizations can achieve more predictable outcomes, higher quality products, and better management of risks.

Disadvantages:

Time-Consuming and Resource-Intensive	Achieving higher maturity levels requires significant time and resource investment, which may be burdensome for organizations with limited capacity.
Rigidity and Overhead	The structured nature of CMM can be perceived as inflexible, leading to unnecessary bureaucracy and

	overhead, especially for smaller organizations or projects with unique needs.

These advantages and disadvantages should be considered when deciding whether to adopt the CMM approach for process improvement.

2.10.

Historically, the introduction of technology has caused profound changes in the labor market and, temporarily at least, displaced people from jobs. Discuss whether the introduction of extensive process automation is likely to have the same consequences for software engineers. If you don't think it will, explain why not. If you think that it will reduce job opportunities, is it ethical for the engineers affected to passively or actively resist the introduction of this technology?

Solution :

The introduction of extensive process automation has historically led to job displacement in many industries. This can happen when technology improves productivity, making certain roles obsolete or reducing the demand for human labor. When considering the potential impact of process automation on software engineers, the situation is complex and multifaceted.

Likely Consequences for Software Engineers:

1. **Reduced Demand for Routine Tasks**:

   Process automation in software engineering could potentially replace more repetitive and routine tasks, such as code generation, testing, debugging, or simple code refactoring. Tools that automate these tasks might reduce the demand for engineers to handle low-level coding activities.

2. **Increased Focus on High-Value Work**:

   While routine tasks may be automated, software engineers will likely shift towards higher-level tasks that require human creativity, problem-solving, and strategic thinking. These could include designing systems, architecting complex software solutions, ensuring security, and collaborating on large-scale projects. Automation is unlikely to completely replace engineers, but rather, change the nature of their work.

3. **Job Evolution**:

   Rather than reducing job opportunities, automation might create new roles for engineers, such as those who specialize in building, maintaining, and optimizing automation tools, or those who focus on higher-level design and

system integration. The demand for engineers with specialized skills in artificial intelligence (AI), machine learning, and automation tools may increase.

Ethical Considerations Regarding Resistance:

If automation leads to reduced job opportunities for software engineers, the ethical question of whether affected engineers should resist the technology arises. Here's how we can look at this issue:

1. **Ethical Resistance to Technology**:
   - **Active Resistance**:

     Actively resisting the introduction of automation might be seen as unethical if the overall benefit of automation leads to improved efficiency, cost reduction, or quality that benefits society at large. However, if engineers resist out of concern for their livelihoods, it is understandable, especially if they lack alternative job opportunities.

   - **Passive Resistance**:

     Engineers might also passively resist by delaying the adoption of automation, but this could hold back innovation and the progress of the field. The role of engineers could then shift to ensuring that new technologies are used ethically and with consideration of their broader societal impacts.

2. **Alternative Ethical Approaches**:
   - **Upskilling and Reskilling**:

     Rather than resisting automation, engineers can focus on upskilling and adapting to new roles. This would involve learning how to work with or create automation tools, ensuring that they remain relevant in the changing job market.

   - **Advocating for Ethical Automation**:

     Engineers could work to ensure that automation technologies are introduced in ways that are ethically responsible, ensuring that displaced workers are supported through retraining programs or that new job opportunities are created.

3. **Balancing Innovation and Workforce Impact**:

   It's crucial to weigh the benefits of automation—such as increased productivity and innovation—against its impact on employment. A

balance needs to be struck where the positive effects of automation are maximized without leaving workers behind. This is an area where software engineers themselves can contribute, using their expertise to shape automation in ways that provide societal benefits while addressing the human cost of job displacement.

Chapter : 3

Exercises

3.1. At the end of their study program, students in a software engineering course are typically expected to complete a major project. Explain how the agile methodology may be very useful for the students to use in this case.

Solution :

The Agile methodology can be highly beneficial for students completing a major project in software engineering. Here's how Agile can help:

Agile Feature	How It Benefits Students' Major Project
Incremental Development	Agile promotes breaking down the project into small, manageable parts, allowing students to build the project step-by-step. This makes it easier to handle complex software and reduces the risk of failure.
Frequent Feedback	Regular iterations and reviews allow students to get frequent feedback from instructors or peers. This helps them adjust their project as they progress, ensuring it meets requirements and standards.
Flexibility	Agile's emphasis on adapting to change means students can modify their project's direction if they encounter challenges or receive new requirements, making it more realistic and adaptable.
Team Collaboration	Agile encourages teamwork and collaboration. This is especially useful for group projects, where students learn to work together, communicate effectively, and assign roles based on strengths.
Time Management	The time-boxed sprints in Agile help students set clear deadlines for each part of their project, helping them manage their time effectively and avoid last-minute rushes.
Continuous Improvement	The Agile process of retrospectives after each sprint allows students to reflect on what went well and what could improve, fostering personal growth and better project outcomes.

Risk Reduction	By delivering parts of the project incrementally and testing each segment, Agile helps students identify and address issues early, reducing the overall risk of project failure.

In summary, Agile provides a structured yet flexible approach that supports iterative progress, collaboration, and continuous improvement, making it ideal for students navigating the complexities of a major software project.

Ex :3.2.
Explain how the principles underlying agile methods lead to the accelerated development and deployment of software.

Solution :

The principles underlying Agile methods drive accelerated development and deployment of software by fostering a dynamic and responsive approach. Here's how Agile principles contribute to speed and efficiency:

Agile Principle	How It Accelerates Development and Deployment
Customer Collaboration Over Contract Negotiation	By prioritizing active collaboration with customers, Agile teams can understand and incorporate client needs from the beginning, reducing the risk of rework and enabling quicker adjustments to requirements.
Working Software Over Comprehensive Documentation	Agile emphasizes delivering functional software early rather than exhaustive documentation. This allows teams to focus on developing and deploying usable components faster, enabling stakeholders to see tangible progress.
Responding to Change Over Following a Plan	Agile promotes adaptability to changing requirements, allowing teams to quickly pivot without being bogged down by rigid plans. This flexibility reduces time lost to revising outdated project plans and accelerates development cycles.
Frequent Delivery of Small, Incremental Releases	Agile's focus on delivering small, functional increments ensures that software components are released and tested regularly. This continuous deployment model allows for rapid improvements and shortens the feedback loop, resulting in faster releases.
Continuous Feedback and Improvement	Regular feedback from stakeholders, through sprints or iterations, allows teams to identify and address issues early. This reduces the need for extensive debugging and rework later, speeding up development overall.
Self-Organizing Teams	Agile encourages empowered teams that can make quick decisions and coordinate without excessive oversight. This autonomy leads to faster problem-solving and efficient workflows.

Sustainable Pace	Agile promotes a manageable, steady work rhythm, reducing burnout and sustaining productivity. By avoiding intense, short bursts of work, teams maintain consistent progress, leading to faster, more reliable delivery.
Focus on Technical Excellence and Simplicity	Agile emphasizes clean, maintainable code and simple solutions, reducing the complexity of the project. This leads to easier updates and faster development cycles by avoiding overengineering.

In essence, Agile principles focus on delivering small, functional parts of the software quickly, adapting to changes, and maintaining close communication with stakeholders. This streamlined, iterative approach significantly accelerates both development and deployment.

Ex : 3.3.

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.

Solution :

Using user stories on cards, as practiced in Extreme Programming (XP), has distinct advantages and disadvantages in requirements description. Here's an overview:

Aspect	Advantages	Disadvantages
Simplicity	User stories on cards are easy to create and understand, focusing on what the user needs without technical jargon.	Cards may oversimplify complex requirements, leading to a lack of detailed information needed for complex systems.
Focus on User Needs	Writing stories from the user's perspective keeps the team focused on delivering value directly aligned with user needs.	Limited focus on non-functional requirements (e.g., security, performance) as cards are typically user-focused and may miss technical details.
Prioritization and Flexibility	Cards can be easily organized, prioritized, or modified, allowing teams to adapt quickly to changing requirements.	Frequent changes and reorganization can lead to scope creep if requirements continually evolve without structured control.
Encourages Collaboration	Stories on cards promote discussions between developers and stakeholders, fostering a collaborative environment.	Over-reliance on verbal clarification can lead to misunderstandings if communication between team members is inconsistent.

Supports Incremental Development	Stories help in breaking down the project into manageable tasks, allowing for iterative development and quicker delivery of usable features.	Breaking down into stories may fragment the project, leading to challenges in maintaining a cohesive view of the system.
Tangible and Visual	Physical cards provide a tangible and visual tool for tracking progress, which can be motivating and easy to reference during team meetings.	Physical cards may be challenging to manage for large, distributed teams, though digital equivalents can mitigate this issue.
Easy to Track Progress	The simplicity of moving cards through stages (e.g., backlog, in-progress, completed) provides a clear visual representation of progress.	Basic cards may not capture the detailed status or dependencies between stories, limiting comprehensive tracking.

In summary, user stories on cards in XP promote simplicity, user-centered focus, and adaptability, supporting collaborative and incremental development. However, they may lack the depth required for complex systems, potentially overlooking technical requirements and dependencies.

Ex :3.4.

In test-first development, tests are written before the code. Explain how the test suite may compromise the quality of the software system being developed.

Solution :

In test-first development, writing tests before code can potentially compromise software quality in a few ways:

Limited Scope of Tests:

If initial test cases are too narrowly defined or lack comprehensive coverage, developers may focus on meeting these specific tests without considering edge cases or broader usage scenarios. This can result in software that works in limited conditions but may fail in real-world applications.

Incomplete Requirements Understanding:

Developing tests early requires a clear understanding of the requirements, which may not be fully formed at the start of the project. This can lead to tests based on assumptions or incomplete

information, resulting in software that ultimately doesn’t meet user expectations or intended functionality.

Overemphasis on Passing Tests:

A focus on passing tests can sometimes overshadow the need to address the functionality's full intent. Developers might create code that technically meets the test requirements without considering factors like robustness, scalability, or usability.

High Test Maintenance:

In a dynamic project, where requirements and code frequently change, test maintenance can become demanding. This may shift developers' focus from enhancing code quality to simply adjusting tests to pass with the latest changes, detracting from time spent on effective development.

Neglect of Non-Functional Requirements:

Test-first development often focuses on functional tests, which may overlook non-functional aspects like performance, security, or scalability. Without early tests for these areas, the software may meet functional requirements but fall short on broader quality attributes.

Tunnel Vision from Early Test Bias:

Writing tests before the code can lead developers to focus too narrowly on meeting the tests, limiting creativity or alternative solutions. This “tunnel vision” effect may prevent the development of more optimized or flexible code solutions.

False Sense of Security:

When all tests pass, developers may assume the software is high-quality. However, if the test suite is not comprehensive or rigorous enough, passing tests can mask potential issues, creating a false sense of security about the software’s robustness.

Overall, while test-first development promotes structured design and early validation, it can compromise quality if it leads to narrow test coverage, places excessive focus on passing tests, or doesn’t account for broader system requirements.

Ex : 3.5.

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

Solution :

Pair programming, where two programmers work together on the same code, often results in a productivity rate that is more than half of what two individual programmers might achieve working separately. Here are four reasons why:

1. **Improved Code Quality and Fewer Errors**:

   With two people continuously reviewing code as it is written, errors and bugs are identified and resolved more quickly. This reduces the time spent on debugging and refactoring later, improving the overall productivity of the pair.

2. **Knowledge Sharing and Skill Complementation**:

   Pair programming enables knowledge transfer between partners, as each programmer brings unique skills, experiences, and perspectives. This shared expertise allows them to solve problems faster and reduces time spent on researching or troubleshooting.

3. **Focused Work and Reduced Distractions**:

   Working with a partner can help both programmers stay focused on the task, as they are more accountable to each other. This minimizes distractions and interruptions, increasing the efficiency of the programming session.

4. **Enhanced Problem-Solving and Faster Decision-Making**:

   With two minds working together, complex problems can be tackled more effectively, and decisions can be made more quickly. The pair can brainstorm solutions, evaluate ideas in real time, and avoid indecision or analysis paralysis, leading to a faster development pace.

In summary, the collaborative nature of pair programming leads to faster error detection, enhanced focus, shared knowledge, and quicker problem-solving, which collectively boosts productivity beyond half of what two individual programmers would achieve.

Ex : 3.6.

Compare and contrast the Scrum approach to project management with conventional plan-based approaches as discussed in Chapter 23. Your comparison should be based on the effectiveness of each approach for planning the allocation of people to projects, estimating the cost of projects, maintaining team cohesion, and managing changes in project team membership.

Solution :

Scrum and conventional plan-based project management approaches differ significantly in their methods and effectiveness regarding team allocation, cost estimation, team

cohesion, and handling changes in project membership. Here's a breakdown of these aspects:

1. **Allocation of People**
   Scrum teams are self-organizing and adjust roles as needed, providing flexibility in dynamic projects. This can be challenging if specific skills are required at certain stages.
   Plan-based approaches assign roles upfront based on a detailed project plan, making them effective for projects with clear, stable requirements.
2. **Cost Estimation**
   Scrum uses flexible, sprint-based estimations like story points, which adapt over time but may lack precision for long-term budgets.
   Plan-based methods use detailed upfront budgeting based on a fixed project scope, which is precise but less adaptable to changing requirements.
3. **Team Cohesion**
   Scrum's daily stand-ups and retrospectives foster close teamwork and communication, supporting strong cohesion.
   Plan-based approaches, with structured roles and less frequent updates, can lead to isolated roles and weaker team bonds.
4. **Managing Changes in Team Membership**
   Scrum's reliance on team dynamics can make frequent membership changes disruptive. New members rely on team support for integration.
   Plan-based projects often have detailed documentation and defined roles, which makes onboarding easier and minimizes disruption.

In summary, Scrum is effective for flexible, cohesive teamwork in adaptive environments, while plan-based methods suit stable projects with clear roles and cost predictability.

Ex : 3.7.

To reduce costs and the environmental impact of commuting, your company decides to close a number of offices and to provide support for staff to work from home. However, the senior management who introduce the policy are unaware that software is developed using Scrum. Explain how you could use technology to support Scrum in a distributed environment to make this possible. What problems are you likely to encounter using this approach?

Solution :

To support Scrum in a distributed environment, technology can facilitate communication, collaboration, and project management. Key tools include:

Tools to Support Distributed Scrum

Technology/Tool	Use Case
Video Conferencing (e.g., Zoom, Teams)	Enables remote Scrum ceremonies (Daily Standups, Sprint Planning, etc.) with face-to-face interaction.
Messaging Tools (e.g., Slack, Teams)	Facilitates real-time communication for updates, questions, and team discussions.
Project Management Tools (e.g., Jira, Trello)	Manages tasks, tracks progress, and provides visibility on user stories and sprints.
Version Control (e.g., Git, GitHub)	Supports distributed code collaboration and version tracking.
CI/CD Tools (e.g., Jenkins, CircleCI)	Automates testing, builds, and deployment, ensuring consistency across locations.
Document Collaboration (e.g., Confluence, Google Docs)	Allows collaborative document creation and sharing.
Time Zone Tools (e.g., World Time Buddy)	Helps schedule meetings across different time zones.
Remote Assistance Tools (e.g., TeamViewer)	Enables collaborative debugging and code walkthroughs.

Challenges and Solutions

Problem	Explanation	Solutions
Time Zone Differences	Synchronous meetings are challenging with team members in different time zones.	Use time zone management tools; stagger meeting times.
Communication Issues	Lack of face-to-face interaction can reduce clarity and team engagement.	Encourage video calls and over-communicate as needed.
Team Cohesion	Building relationships and maintaining engagement can be harder remotely.	Virtual team-building activities; frequent check-ins.
Alignment on Priorities	Remote teams might struggle to stay aligned on priorities.	Use clear communication via project tools; regular check-ins.
Technical Problems	Remote tools may face technical issues, impacting productivity.	Ensure strong technical support and backup plans.
Focus and Motivation	Working from home may cause distractions and reduce motivation.	Set clear expectations; maintain frequent check-ins.

Conclusion:

Technology enables effective remote Scrum through communication, collaboration, and project management tools. However, challenges like time zone differences and communication issues need careful management for successful implementation.

Ex :3.8.

Why is it necessary to introduce some methods and documentation from plan-based approaches when scaling agile methods to larger projects that are developed by distributed development teams**?**

Answer :

When scaling agile methods to larger projects with distributed teams, some plan-based practices are necessary for the following reasons:

1. Coordinatio**n:** Agile works best with small, co-located teams. Plan-based methods help align and synchronize efforts across multiple, remote teams.
2. **Clear Communication:** Agile favors informal communication, but distributed teams need structured documentation to ensure clarity and set expectations.
3. **Managing Complexity:** Larger projects often have complex dependencies. Planning tools (e.g., Gantt charts) help manage these effectively.
4. **Consistency:** Documentation ensures consistency in design, quality assurance, and integration across teams.
5. **Risk Management:** Plan-based approaches help identify and mitigate risks early, especially in large projects with distributed teams.
6. **Long-Term Vision:** Agile focuses on short-term goals. Plan-based methods provide a high-level roadmap to guide long-term objectives.
7. **Compliance:** Some industries require detailed documentation for regulatory compliance, which agile alone may not address.
8. **Stakeholder Engagement:** Plan-based methods help structure regular stakeholder communication and reporting, ensuring consistent updates across distributed teams.

In essence, combining agile with plan-based practices provides necessary structure, alignment, and consistency in large-scale, distributed projects.

Ex : 3.9.

Explain why agile methods may not work well in organizations that have teams with a wide range of skills and abilities and well-established processes.

Solution :

1 **Varying Skill Levels**

Agile assumes that team members are relatively self-sufficient and can

collaborate closely with minimal supervision. Teams with varying skill levels may struggle to work in an agile environment, where members are expected to take on multiple roles and contribute to all stages of development. Less experienced members may need more guidance, making the flexible, fast-paced nature of agile difficult to manage.

2 **Steep Learning Curve**
Agile requires all team members to be well-versed in agile principles, practices, and tools. In organizations with a wide skill range, the learning curve for agile practices can be steep, especially for those unfamiliar with agile frameworks. Teams may struggle to adapt to agile's collaborative, iterative, and self-organizing processes.

3 **Resistance to Change**
Agile thrives on adaptability and continuous improvement. In organizations with well-established processes, there may be significant resistance to change. Employees accustomed to traditional workflows may find it hard to transition to agile's iterative, flexible approach, leading to friction and inefficiencies.

4 **Difficulty in Cross-functional Collaboration**
Agile teams are cross-functional, meaning that members should ideally have diverse skills and be able to collaborate closely on all tasks. In organizations with specialized teams (e.g., developers, testers, designers), it may be difficult to break down silos and foster collaboration. The need for everyone to be a "jack of all trades" in an agile team may not align with team members' expertise.

5 **Lack of Well-Defined Roles**
Agile encourages flexible roles and team members wearing multiple hats. In organizations with established roles, employees might be reluctant to shift responsibilities or take on tasks outside their expertise. This can hinder the flexibility needed for agile to succeed.

6 **Mismatch with Rigid Processes**
Agile promotes adaptive, iterative work with minimal formal documentation and rigid structures. Organizations with well-defined, rigid processes may struggle to integrate agile. Their focus on standardized procedures, strict documentation, and hierarchical decision-making may clash with agile’s emphasis on flexibility and empowerment of self-organizing teams.

7 **Inadequate Infrastructure for Agile**
Agile requires supportive infrastructure, including tools for collaboration, continuous integration, and automated testing. In organizations with established processes, legacy systems or outdated tools may not support agile workflows effectively, requiring significant changes to infrastructure.

8 **Management and Accountability Issues**
Agile relies on empowered teams that self-organize and make decisions. In organizations with well-established processes, managers may be accustomed to

a top-down approach. Agile's decentralized decision-making can create confusion around accountability and authority, leading to resistance from leadership.

Ex : 3.10.

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**.**

Solution :

To avoid the problem of a user "going native" and losing sight of the needs of their colleagues, here are three strategies:

1. Regular Feedback from Other Users

- **Advantages**: Ensures that the user remains aligned with broader needs and provides early issue detection.
- **Disadvantages**: Can slow down development and create friction if feedback conflicts with the user's perspective.

2. Rotating Users or Representatives

- **Advantages**: Brings diverse user perspectives, reducing bias and broadening feedback.
- **Disadvantages**: Can lead to inconsistencies and delays as new users need time to catch up.

3. Clear Separation of Roles

- **Advantages**: Keeps the user focused on needs rather than technical details, preventing them from adopting the development team's mindset.
- **Disadvantages**: May lead to frustration or unrealistic expectations due to lack of technical understanding.

Summary of Approaches:

Approach	Advantages	Disadvantages
Regular Feedback from Other Users	Aligned with broader needs, early issue detection	Can slow down progress, potential conflicts
Rotating Users or Representatives	Diverse insights, reduces bias	Inconsistencies, delays as new users adjust
Clear Separation of Roles	Focuses user on needs, avoids adopting development mindset	Disconnect between user and team, unrealistic expectations

Chapter : 4

Exercises :

Ex : 4.1.

Identify and briefly describe four types of requirements that may be defined for a computerbased system.

Solution :

Here are four types of requirements typically defined for a computer-based system:

Requirement Type	Description
Functional Requirements	Define what the system should do, including the specific behaviors, tasks, or functionalities. Example: "The system must allow users to log in using a username and password."
Non-Functional Requirements	Describe the quality attributes, performance criteria, or constraints of the system. Example: "The system should handle 1,000 concurrent users with a response time of less than 2 seconds."
Technical Requirements	Specify the technology, platforms, or tools the system should use or support. Example: "The system must be developed in Python and run on Linux servers."
Business Requirements	Capture the high-level goals or needs of the organization that the system must satisfy. Example: "The system must reduce order processing time by 20% within six months."

Each type serves a distinct purpose in defining the scope, design, and implementation of the system.

Ex : 4.2.

Discover ambiguities or omissions in the following statement of the requirements for part of a drone system intended for search and recovery:

The drone, a quad chopper, will be very useful in search and recovery operations, especially in remote areas or in extreme weather conditions. It will click high-resolution images. It will fly according to a path preset by a ground operator, but will be able to

avoid obstacles on its own, returning to its original path whenever possible. The drone will also be able to identify various objects and match them to the target it is looking for.

Solution :

Here are some potential ambiguities and omissions in the provided requirements for the drone system:

Aspect	Ambiguity or Omission
"Very useful"	The term "very useful" is vague. What specific capabilities or metrics define "usefulness" for search and recovery operations?
"High-resolution images"	The term "high-resolution" is subjective. What resolution (e.g., megapixels) or quality standards are required?
"Extreme weather conditions"	There is no specification of the types of weather conditions (e.g., temperature range, wind speed, precipitation) the drone must withstand.
"Avoid obstacles"	The requirement lacks detail on how obstacles are detected, the distance at which avoidance is triggered, and the types of obstacles it can recognize (e.g., trees, power lines).
"Identify various objects"	What types of objects should the drone identify? Is there a predefined list or general criteria for identification? How accurate must the identification be?
"Match them to the target"	There is no detail about how targets are defined, how matching works (e.g., by color, shape, size), or the acceptable error margin in matching.
"Preset by a ground operator"	How is the path preset? Is it via software, a manual controller, or another interface? What happens if the connection to the operator is lost?
Power and endurance	There is no mention of the drone’s battery life, operational range, or the expected duration of a single flight mission.
Safety and compliance	There is no mention of safety protocols (e.g., emergency landing procedures) or compliance with aviation regulations.

Addressing these ambiguities and omissions would make the requirements clearer, measurable, and actionable.

Ex : 4.3.

Rewrite the above description using the structured approach described in this chapter. Resolve the identified ambiguities in a sensible way.

Solution :

Structured Rewrite

Function: Search and Recovery Drone

Description:

A drone used for search and recovery, capable of identifying a target along a preset path by capturing and analyzing images.

Inputs: Target image and preset path.

Source: Ground operator.

Outputs: High-resolution images stored and organized by relevance to the target.

Objective:

The drone shall navigate the preset path, capture images, and match them with the target image.

Conditions:

- **Precondition:** Target image and path must be provided.
- **Postcondition:** Target found or not found.

Reactions:

- Target not found: Drone completes the route and informs the operator.
- Obstacle avoidance: Drone reroutes and attempts to reconnect to the original path.
- Route changes: Operator can update the route mid-flight.

Ex :4.4.

Write a set of non-functional requirements for the drone system, setting out its expected safety and response time.

Solution :

Non-Functional Requirements for Drone System

1. Safety Requirements

- **Collision Avoidance**: The drone must detect and avoid obstacles within 5 meters, rerouting or landing safely.
-
- **Emergency Response**: The drone must initiate an emergency return or land within 30 seconds in case of failure.

- **Weather Resistance**: The drone must operate in winds up to 15 m/s and temperatures between -10°C to 45°C.
- **Data Encryption**: All communications must use AES-256 encryption.

2. Performance Requirements

- **Response Time**: The drone must respond to commands within 2 seconds.
- **Image Capture**: Images must be captured and processed every 5 seconds.
- **Target Detection**: The drone must confirm target detection within 10 seconds.
- **Battery Life**: The drone must operate for at least 60 minutes per mission.

3. Reliability

- **Uptime**: The system must have 99.5% uptime over a month.
- **Failure Recovery**: The drone must recover from minor software failures automatically.

Ex : 4.5.

Using the technique suggested here, where natural language descriptions are presented in a standard format, write plausible user requirements for the following functions:

An unattended petrol (gas) pump system that includes a credit card reader. The customer swipes the card through the reader, then specifies the amount of fuel required. The fuel is delivered and the customer's account debited.

The cash-dispensing function in a bank ATM.

In an Internet banking system, a facility that allows customers to transfer funds from one account held with the bank to another account with the same bank.

Solution :

User Requirements for Unattended Petrol Pump System

- **Function**: Fuel Dispensing
  - **Description**: The system shall allow customers to swipe a credit card, specify the amount of fuel required, and automatically dispense the fuel while debiting the specified amount from the customer's account.
  - **Precondition**: The customer must have a valid credit card with sufficient funds.
  - **Postcondition**: The specified amount of fuel is delivered, and the customer's account is debited accordingly.
  - **Response**: The system shall notify the customer upon successful transaction completion.

User Requirements for Cash Dispensing Function (ATM)

User Requirements for Fund Transfer (Internet Banking)

- **Function**: Fund Transfer

- **Description**: The system shall allow customers to transfer funds between accounts held with the same bank by entering transfer details and authorizing the transaction.
- **Precondition**: The customer must be logged in with valid credentials and have sufficient funds in the source account.
- **Postcondition**: The transfer is completed, and the accounts are updated accordingly.
- **Response**: The system shall notify the customer upon successful completion of the transfer.

These are concise user requirements for each function, following a standard format with clear descriptions, conditions, and expected responses.

Ex : 4.6.

Suggest how an engineer responsible for drawing up a system requirements specification might keep track of the relationships between functional and non-functional requirements.

Solution :

An engineer can track the relationships between functional and non-functional requirements by using a requirements traceability matrix. This tool links functional requirements (what the system should do) with non-functional requirements (how the system should perform) to ensure all aspects are addressed. The matrix can be updated during the project lifecycle to monitor dependencies and changes, helping to maintain alignment between the two types of requirements

Ex : 4.7.

Using your knowledge of how an ATM is used, develop a set of use cases that could serve as a basis for understanding the requirements for an ATM system.

Solution :

Here are some use cases for an ATM system, representing different scenarios and interactions between the user and the system:

Use Case ID	Use Case Title	Primary Actor	Description	Preconditions	Postconditions
UC1	Withdraw Cash	Customer	The customer inserts a card, enters PIN, selects withdrawal amount, and receives cash.	Customer has a valid card and PIN	Customer receives cash and receipt

UC2	Check Account Balance	Customer	The customer inserts a card, enters PIN, and views the current account balance.	Customer has a valid card and PIN	Customer views account balance
UC3	Transfer Funds	Customer	The customer selects transfer option, enters recipient details, and transfers funds between accounts.	Customer has sufficient funds	Funds are transferred between accounts
UC4	Deposit Cash	Customer	The customer inserts a card, enters PIN, selects deposit option, and deposits cash into the account.	Customer has a valid card and PIN	Cash is deposited into the account
UC5	Change PIN	Customer	The customer selects the change PIN option, enters the old PIN, and sets a new one.	Customer has a valid card and PIN	Customer's PIN is updated
UC6	Print Mini Statement	Customer	The customer inserts a card, enters PIN, and requests a mini statement of recent transactions.	Customer has a valid card and PIN	Customer receives mini statement
UC7	ATM Maintenance	Bank Technician	The technician performs maintenance tasks like refilling cash, fixing hardware, or software updates.	ATM is in maintenance mode	ATM is serviced and operational again

These use cases help identify the key functions that the ATM should support and serve as a foundation for understanding system requirements.

Ex :4.8.

To minimize mistakes during a requirements review, an organization decides to allocate two scribes to document the review session. Explain how this can be done.

Solution : To minimize mistakes during a requirements review, two scribes can be allocated as follows:

1. **Divide Tasks:**
   One scribe records main discussion points, while the second focuses on detailed examples and clarifications. This ensures both high-level and detailed information is covered.
2. **Collaborate:**
   The scribes compare notes during the session to ensure no key points are missed and to maintain consistency in the documentation.
3. **Double-Check:**
   After the session, both scribes review the notes together to verify accuracy and ensure no contradictions or missing details.
4. **Clarify:**
   If any points are unclear, scribes ask participants for clarification to avoid misinterpretation and ensure the documentation is correct.
5. **Consolidate:**
   The scribes combine their notes into a single document, reviewing it for completeness and accuracy before sharing it with the team.

Ex :4.9.

When emergency changes have to be made to systems, the system software may have to be modified before changes to the requirements have been approved. Suggest a model of a process for making these modifications that will ensure that the requirements document and the system implementation do not become inconsistent.

Solution :

A **Change Control Process** can be used to ensure consistency between the requirements document and the system implementation:

1. **Identify Emergency Change:**
   When an urgent change is needed, a temporary change request is submitted, outlining the modification and its urgency.
2. **Assess Impact:**
   The impact of the change on existing requirements is assessed by both technical and requirements teams.
3. **Implement Change**:
   The system software is modified to address the emergency, but with temporary deviations, if necessary.
4. **Update Requirements:**
   The requirements document is updated as soon as possible to reflect the emergency change, ensuring alignment.
5. **Review and Approve:**
   Once the emergency change is implemented, the updated requirements are formally reviewed and approved, and the system is tested against them.

This process ensures that the system and the requirements remain consistent despite emergency changes.

Ex : 4.10.

You have taken a job with a software user who has contracted your previous employer to develop a system for them. You discover that your company's interpretation of the requirements is different from the interpretation taken by your previous employer. Discuss what you should do in such a situation. You know that the costs to your current employer will increase if the ambiguities are not resolved. However, you also have a responsibility of confidentiality to your previous employer.

Solution :

In this situation, you should:

1. **Review requirements:**

   Understand both interpretations of the requirements to identify ambiguities.

2. **Consult current employer:**

   Inform them of the issue and potential cost increases without disclosing confidential details.

3. **Clarify with the client:**

   Engage the client directly to resolve the ambiguity, ensuring no sensitive information is shared.

4. **Maintain confidentiality:**

   Avoid revealing proprietary details from your previous employer.

5. **Consider a neutral third party**:

   If necessary, suggest involving a mediator to resolve differing interpretations.

6. **Document everything:**

   Keep detailed records of discussions and decisions for future reference.