1. Software engineering is the systematic application of engineering approaches to the development, operation, and maintenance of software. It involves designing, building, testing, and maintaining software systems to ensure they meet quality, performance, and reliability standards. Unlike traditional programming, which focuses primarily on writing code to solve specific problems, software engineering encompasses the entire software development process, including requirements analysis, design, implementation, testing, deployment, and maintenance *(Pressman, 2014)*.
2. The Software Development Life Cycle (SDLC) is a process used by software development teams to design, develop, and maintain high-quality software (Sommerville, 2015).

* **Requirements Gathering:** In this phase, stakeholders' requirements are collected and analysed to understand the scope and objectives of the project.
* **Design:** During this phase, the system architecture, data structures, and interfaces are designed based on the gathered requirements.
* **Implementation:** This phase involves writing code according to the design specifications. It includes coding, unit testing, and debugging.
* **Testing:** In this phase, the software is tested to ensure it meets the specified requirements and is free of defects.

1. The Agile and Waterfall models are two different approaches to software development:

The Waterfall model preferred in scenarios where requirements are well-defined, because its structured (Highsmith, 2002). Agile is best in dynamic environments where requirements are likely to evolve.

* **Agile Model:** Agile is an iterative and flexible approach to software development that emphasizes collaboration, adaptability, and customer feedback. It breaks the development process into small, incremental cycles called sprints, allowing for frequent inspection and adaptation of the product (Highsmith, 2002).
* **Waterfall Model:** In the Waterfall model, the development process is divided into sequential phases, with each phase dependent on the deliverables of the previous phase (Highsmith, 2002). It follows a rigid, linear approach where requirements are defined upfront, and changes are difficult to accommodate once the development process begins.

1. Requirements engineering is the process of eliciting, analysing, documenting, and managing requirements throughout the software development lifecycle. It involves understanding stakeholders' needs and translating them into specific, actionable requirements that guide the development process (Weigers, 2013). Requirements engineering ensures that the software meets the desired functionality, performance, and quality standards while also managing changes and conflicts that may arise during development.
2. Modularity in software design refers to the practice of breaking down a system into smaller, self-contained modules or components, each responsible for a specific function or feature. Modularity improves maintainability and scalability by allowing developers to isolate changes to individual modules without affecting the entire system (Martin, 2005). It also facilitates code reuse and enhances collaboration among team members.
3. Software testing involves evaluating a software system or its components to ensure that they meet specified requirements and quality standards (Meyers, 2011). The levels of software testing include:

* **Unit Testing:** Testing individual units or components of the software in isolation to verify their correctness.
* **Integration Testing:** Testing the interaction between different modules or components to ensure they work together as intended.
* **System Testing:** Testing the entire system as a whole to validate its functionality, performance, and reliability.
* **Acceptance Testing:** Testing conducted by end-users or stakeholders to determine whether the software meets their requirements and expectations.
* Testing is crucial in software development because it helps identify defects early in the development process, reduces the risk of software failures in production, and ensures the quality and reliability of the final product.

1. Version control systems (VCS) are software tools that track and manage changes to source code and other files. They allow developers to collaborate effectively, track revisions, revert to previous versions, and manage concurrent changes to the same files (Loeliger, 2012). Examples of popular version control systems include Git, Subversion (SVN), and Mercurial. Many of these systems offer features such as branching, merging, and conflict resolution, which are essential for managing complex software projects.
2. Software Project Manager, oversees the planning, execution, and delivery of software projects. Responsibilities include: setting project goals, allocating resources, managing timelines, and mitigating risks. Some challenges are: balancing scope, time, and resources, handling changing requirements, and ensuring stakeholder satisfaction (Schwalbe, 2015).
3. Software maintenance defined is the process of updating, enhancing, and fixing issues in software after it's deployed. The methods are: fixing bugs, adapting to changes in the environment, improving performance or usability, proactive measures to reduce future issues. Ensuring the software remains functional, secure, and aligned with evolving needs is essential in its maintenance (Lehman, 2003).
4. Issues with ethical considerations in software engineering: privacy violations, bias in algorithms, software piracy, lack of transparency.   
   Adherence includes: following professional codes of conduct, considering the societal impact of software, being transparent about decisions and actions.

# References

* Highsmith, J. (2002). Agile Software Development Ecosystems. Addison-Wesley
* Johnson, D. G., & Nissenbaum, H. (1995). Computers, Ethics, and Social Values. Prentice Hall.
* Lehman, M. M., & Ramil, J. F. (2003). Software Evolution and Feedback: Theory and Practice. Wiley.
* Loeliger, J., & McCullough, M. (2012). Version Control with Git: Powerful Tools and Techniques for Collaborative Software Development. O'Reilly Media.
* Martin, R. C. (2009). Clean Code: A Handbook of Agile Software Craftsmanship. Prentice Hall.
* Myers, G. J., Sandler, C., & Badgett, T. (2011). The Art of Software Testing (3rd ed.). Wiley.
* Pressman, R. S. (2014). Software Engineering: A Practitioner's Approach. McGraw-Hill Education.
* Schwalbe, K. (2015). Information Technology Project Management (8th ed.). Cengage Learning.
* Sommerville, I. (2015). Software Engineering (10th ed.). Pearson Education Limited.
* Wiegers, K. E., & Beatty, J. (2013). Software Requirements (3rd ed.). Microsoft Press.