**What is software engineering, and how does it differ from traditional programming? Software Development Life Cycle (SDLC):**

Software Engineering - A set of concepts, methods, techniques, tools, systems, etc., designated for ensuring high quality and productivity of engineering activities on creation and operation (i.e., use) of software. (In other words, everything of what persons involved in such activities reasonably should know, should have a skill to do, and should have in disposition, in order to perform the work in a professional way.)(Dzerzhinskiy & Raykov, n.d.).

It is a systematic collection of good program development practices and techniques” (Mall, 2015)

Software engineering is a comprehensive, methodical approach that emphasises quality, cooperation, and formal processes throughout the entire software lifecycle. It covers the full software development and maintenance process. Contrarily, traditional programming focuses primarily on the act of developing code, frequently using a more casual and individualised approach. Comprehending these distinctions is critical for proficiently overseeing and completing software projects, particularly when they expand in intricacy and magnitude.

**Explain the various phases of the Software Development Life Cycle. Provide a brief description of each phase. Agile vs. Waterfall Models:**

Software Development Life Cycle refers to the methodology used to plan, create, test, deploy, and maintain a software throughout its lifecycle until decommission.(Swanlund et al., 2024).

It is a framework for planning, analyzing, designing, developing, testing, and deploying software. There are many different SDLC methodologies available, each with its advantages and disadvantages. The best methodology for a particular project will depend on factors such as the size and complexity of the project, the availability of resources, and the preferences of the project team. (-, 2023)

**Phases of SDLC**:

**Planning**: Specify the needs, goals, and scope of the project. Determine the roles of the stakeholders.Make a project plan that includes resources and timetables.

**Analysis**: Collect and record specific needs from users and stakeholders. Examine the data collected to comprehend the functional and non-functional aspects of the system. Create functional requirements, user stories, or use cases.

**Design**: Develop an architectural design that specifies the interface between the software components. Provide thorough technical specs. Create the user experience (UX) and user interface (UI).

**Implementation (Coding)**: Using the design and specs as a guide, write and test the actual code. Test individual components individually to make sure they function as intended. Assemble code modules together.

**Testing**: Conduct testing at many levels, such as system, integration, and user acceptability testing (UAT). Find and address flaws and problems. Confirm that the programme satisfies the specified specifications.

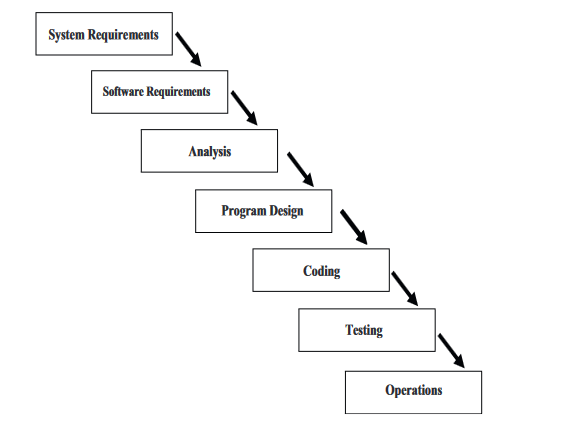
**Deployment**: Set up a staging environment for the programme to be tested and validated one last time. Configure servers and databases in preparation for the production environment. Introduce the programme to clients or end users.

**Maintenance and Support**: Handle and correct any reported problems or faults. Keep an eye on and maintain the programme in the production environment. Updates, improvements, and patches should be applied as needed.

**Compare and contrast the Agile and Waterfall models of software development. What are the key differences, and in what scenarios might each be preferred? Requirements Engineering:**

**Waterfall Model:**

The Waterfall Model focuses mostly on required specifications or high-level design freezing early in the development life-cycle, before more in-depth design and implementation work is undertaken. Therefore, if needs are not clearly understood or stated or are likely to change as the project progresses, the Waterfall paradigm is probably not appropriate. (Casteren, 2017)

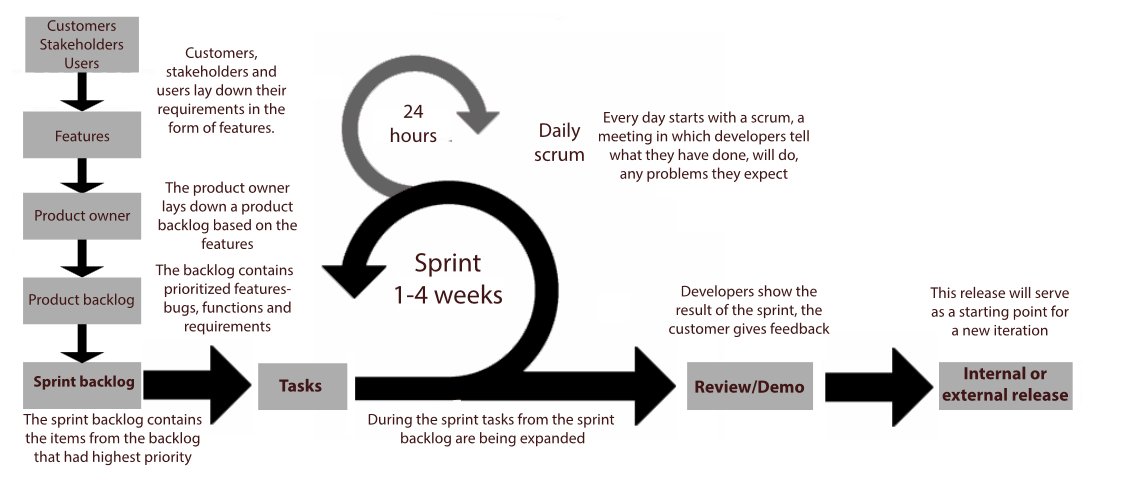


Preferred Scenarios of Waterfall Model:

* Projects with stable and well-defined requirements.
* Regulatory and compliance-driven projects where documentation is critical.
* Environments where a sequential and predictable process is required.

**Scrum Model:**

While agile approaches differ in their methods and approaches, they all have iterative development, a focus on interaction and communication, and a reduction in the number of resource-intensive intermediary artefacts. Agile methodologies additionally incorporate feature planning, dynamic prioritisation, and brief iterative cycles. (Casteren, 2017)



Preferred Scenarios for Agile Model:

* Projects with uncertain or rapidly changing requirements.
* Environments where customer feedback and involvement are crucial.
* Development of innovative or user-focused products.

**Comparison between waterfall and agile methodology** (Freeman, 2022)

Their shared objectives are to create software programmes of the highest calibre and satisfy their customers.

They carry out identical tasks, such as gathering requirements, designing, creating, testing, and implementing.

Planning, implementing the project, and tracking its advancement using the two approaches form the basis of any project.

**What is requirements engineering? Describe the process and its importance in the software development lifecycle. Software Design Principles:**

Requirements engineering (RE) is the process of discovering the purpose for which a software / product is intended, by identifying stakeholders and their needs, and documenting these in a form that is amenable to analysis, communication, and subsequent implementation. (Nuseibeh & Easterbrook, 2000)

**Importance of Requirement Engineering:**

Requirement Engineering is used to develop effective software and in reducing software errors at the early stage of the development of software. Since Requirement Engineering (RE) has great role in different stages of the SDLC, its consideration in software development is crucial.(Chakraborty et al., 2012)

**Process of Requirement Engineering** (Nuseibeh & Easterbrook, 2000)

**Elicitation Requirements**

The activity that is most frequently recognised as the start of the RE process is probably requirement elicitation. It is preferable to use the term "elicitation" rather than "capture" to avoid giving the impression that requirements can be found and gathered by merely asking the proper questions. Before the requirements engineer is comfortable that a comprehensive enough collection of requirements for a system has been gathered, information obtained during requirements elicitation frequently needs to be understood, analysed, modelled, and validated.

**Modelling and Analysing Requirements**

A key component of RE is modelling, which is the creation of abstract descriptions that allow for interpretation. A wide variety of RE process outputs can be represented using models. Additionally, a lot of modelling techniques are employed as elicitation tools, with the generated partial models and modelling notation serving as catalysts for additional data collection.

**Communicating Requirements**

Requirement Engineering is not only a process of discovering and specifying requirements, it is also a process of facilitating effective communication of these requirements among different stakeholders. The way in which requirements are documented plays an important role in ensuring that they can be read, analysed, (re-)written, and validated.

**Agreeing Requireiments**

Keeping all stakeholders in accord during the elicitation and modelling of needs can be challenging, particularly when there are conflicting objectives among the parties. Remember that validation is the process of proving that the models and requirements that were elicited accurately reflect the needs of the stakeholders. A prerequisite for both requirement validation and stakeholder conflict resolution is an explicit description of the requirements.

**Evolving Requirements**

Stakeholder requirements and the environment in which they operate are constant sources of evolution for successful software systems. As a result, one of the core functions of requirement engineering is change management.

**Software Design Principles** (Team, n.d.)

SOLID Principles

The SOLID principles are a set of software design guidelines that help developers create more maintainable, flexible, and robust software systems.

Don’t Repeat Yourself (DRY)

The DRY principle emphasizes the importance of avoiding duplication in code. Duplication can lead to inconsistencies, increased maintenance effort, and a higher likelihood of introducing errors when changes are required.

Encapsulation

Encapsulation is a core concept in object-oriented programming that promotes the bundling of data (attributes) and methods (functions) that operate on the data within a single unit, typically a class.

Principle of Least Astonishment (PoLA)

PoLA is a design guideline that encourages developers to create software components that behave predictably and intuitively, minimizing surprises for users or other developers interacting with the component.

**Explain the concept of modularity in software design. How does it improve maintainability and scalability of software systems? Testing in Software Engineering:**

**Modularity** is a fundamental design principle in software engineering aimed at creating software in a way that minimizes dependencies among the components of a system. This helps to localize the impact of changes, simplifies maintenance, and enhances the understandability of the system.

**How does it improve maintainability and scalability of software systems?**

The need to effectively handle expansion, shifting requirements, and rising user demands becomes critical as applications and systems develop. Software scalability can be a difficult and error-prone task without an effective architectural foundation in place, which can result in problems like bottlenecks, decreased system stability, and exorbitant maintenance costs.

In order to overcome these difficulties, modular software design divides large, complicated systems into smaller, more manageable modules or components. These modules work together harmoniously inside the overall software system, but they also operate independently. This method makes it easier to scale the software and add new features and improvements.(selleo.com, 2024)

**Testing in Software Engineering**

Software testing is an empirical technical investigation conducted to provide stakeholders with information about the quality of the product or service/system under test.(Iqbal, 2017)

**Describe the different levels of software testing (unit testing, integration testing, system testing, acceptance testing). Why is testing crucial in software development? Version Control Systems:**

**Unit Testing** - This testing emphasizes on individual units or modules in isolation. It is a testing in which the smallest testable portion of software is tested to verify its functionality against its specification. The unit can be a constructor or destructor at class level in an object-oriented environment or a structure in procedural programming paradigm. (Umar, 2020)

**Integration Testing** - Involves testing two or more combined units that must work together to ensure an error-free flow of control and data (such as consistency of parameters, file format, and so on) among combined units and their overall correct design and integration. User interface, use-case, interaction, and big bang (integrate and test all modules at once) are some of the integration testing types. (Umar, 2020)

**System Testing** - Involves testing an integrated complete software to check against its compliance with its requirements. It verifies the overall interaction of components to ensure the unanimous working of all modules and programs without error. It involves various types of both functional (tests functionality of software) testing and non-functional (tests quality of software) testing such as performance, reliability, usability, and security testing. (Umar, 2020)

**Acceptance Testing** - This testing is performed to validate the software against customer requirements. This testing is done to ensure that the software does what the customer wants it to do and check the acceptability of the system. User Acceptance Testing (UAT), as sometimes called, comprises of two testing types: Alpha testing: is a testing performed by both development team and users using made-up data, and Beta testing in which users start using the software with real data and carefully observer the software for errors. (Umar, 2020)

**Why is Testing crucial in software development?**

Software testing is really required to point out the defects and errors that were made during the different phases of the SDLC. It is essential since testing makes sure of the Customer’s reliability and their satisfaction in the application. It is very important to ensure the Quality of the product. (*IJSRD - International Journal for Scientific Research & Development| Vol. 6, Issue 12, 2019 | ISSN (Online): 2321-0613*, n.d.)

**What are version control systems, and why are they important in software development? Give examples of popular version control systems and their features.**

**Version Control System**

Version Control System (VCS) is a system that manages the development of an evolving object. In other words, it is a system that records any changes made by the software developers. There are a lot of uses for VCS in software development that makes the development process easier and faster. (Zolkifli et al., 2018)

**Version Control System (VCS) is important for the following reason:**

VCS has been proven to accelerate and simplify the software development process. There are loads of advantages in using VCS for software projects. VCS allows people to work absolutely freely with the team. They can work on any file at any time without overlapping each other’s work by writing over other people’s code. If two developers make changes in the same file, the VCS will merge the changes or warn them that some of the codes are conflicting because VCS has the ability to track each alteration or changes made in the files or codes (Zolkifli et al., 2018)

**Examples of popular version control systems and their features**

**Git** is a distributed VCS that has gained immense popularity among developers. It allows developers to work collaboratively on a project, keep track of changes made to the code, and revert changes if needed. It has a rich command-line interface and can be integrated with many IDEs and text editors. (Božić, 2023)

**Subversion (SVN)** - SVN is a centralized VCS that has been around for many years. It is widely used in enterprise environments and has a good track record for stability and reliability. SVN has a simple command-line interface and can be integrated with many IDEs. (Božić, 2023)

**Mercurial**. Mercurial is a distributed VCS that is similar to Git in many ways. It has a user-friendly command-line interface and can be integrated with many IDEs and text editors. It is popular among Python developers.(Božić, 2023)

**Perforce**. Perforce is a centralized VCS that is widely used in the gaming and entertainment industries. It is known for its scalability and performance and has a rich command-line interface. Perforce can be integrated with many IDEs and text editors. (Božić, 2023)

**Discuss the role of a software project manager. What are some key responsibilities and challenges faced in managing software projects?**

Project managers are responsible for planning, executing, monitoring, managing and completing projects. However, that is just the tip of the iceberg of project management. (HEZAM, 2021)

Here are some of the **Key Responsibilities** of a project manager:

**Create a plan**: Project managers are responsible for planning the actual course of the project. The plan should include the scope of the project, the timeline, and the budget. This can include identifying the right tools for the job. (HEZAM, 2021)

**Join the team**: Finding the right team is essential to the success of the project. All project teams will vary depending on the scope of the action and the tasks required to complete the project. (HEZAM, 2021)

**Assign tasks**: Project managers must provide their team with a clear description of specific tasks and a timeline for all parts of the project. While each team member will be committed to his or her responsibilities, many tasks will require the cooperation of internal and external team members. (HEZAM, 2021)

**Leading the team**: Now that the team is organized and their tasks are assigned, the project manager must keep the machine properly installed. This will include looking at specific people to get status updates, identifying and removing roadblocks, negotiating disagreements, maintaining team ethics, and providing training and counseling. (HEZAM, 2021)

**Budget management**: Most projects will require some expense, which means understanding how to integrate project budget and cost management is critical to success. This will include comparing real-life costs with estimates, and adjusting the project plan if necessary. (HEZAM, 2021)

**Challenges Faced in managing software projects**

**Unestablished Project Infrastructure** - An unestablished project environment is a common software development challenge in terms of its impact on project delivery. If the environment is not available, then there is no way you can proceed with your project on time and under budget.(Thakkar, 2024)

**Keeping Up with Changing Requirements** - A major reason for the complexity of software development projects is the constant changing of requirements. Not surprisingly, 33% of the respondents of the Stack Overflow Developer Survey consider building products with unspecific requirements as their biggest challenge. Requirements gathering is a lot more than a handful of business consultants coming up with their ideal product – it is understanding fully what a project will deliver. (Thakkar, 2024)

**Quality Assurance** - Not reviewing code, or suppressing errors are a means that software developers use to save time and meet deadlines. (Thakkar, 2024)

**Undefined Quality Standards** - Defect identification is inevitable during functionality testing, even if the product has been through thorough unit testing during the development phase. (Thakkar, 2024)

**Define software maintenance and explain the different types of maintenance activities. Why is maintenance an essential part of the software lifecycle? Ethical Considerations in Software Engineering:**

**Software maintenance** is the totality of activities required to provide cost-effective support to a software system. Activities are performed during the pre-delivery stage as well as the post-delivery stage. Pre-delivery activities include planning for post-delivery operations, supportability, and logistics determination. Post-delivery activities include software modification, training, and operating a help desk. (Canfora & Cimitile, 2001)

**Types of Maintenance activities** (Canfora & Cimitile, 2001)

**Corrective maintenance**: reactive modification of a software product performed after delivery to correct discovered faults.

**Adaptive maintenance**: modification of a software product performed after delivery to keep a computer program usable in a changed or changing environment.

**Perfective maintenance**: modification of a software product performed after delivery to improve performance or maintainability.

**Emergency maintenance**: unscheduled corrective maintenance performed to keep a system operational.

**Why is maintenance an essential part of the software lifecycle?**

Without maintenance, any software will be obsolete and essentially useless over time. Creating a new piece of software and launching it into the world is an exciting step for any company. A lot goes into creating your software and its launch including the actual building and coding, licensing models, marketing, and more. However, any great piece of software must be able to adapt to the times. This means monitoring and maintaining properly. As technology is changing at the speed of light, software must keep up with the market changes and demands. (thalesgroup.com, n.d.)

**What are some ethical issues that software engineers might face? How can software engineers ensure they adhere to ethical standards in their work?**

**Ethical Issues that software engineers might face**

**Privacy** – Handling, storing, sharing user data only under the circumstances and for the purposes that the user sets.

**Sustainability** – Energy consumption of the software artifact, caring about energy throughout the Software Engineering process and in the documentation.

**Transparency** – Transparent decision-making procedures of intelligent systems, publicly available ethics politics by software development organizations.

**Diversity** – Gender, race and age distribution of professionals in a development team.

**Work Ethics** – Decisions on which bugs to fix and how quickly, ensuring quality of the code before release.

**Business ethics** – Informing users of a changed business model, including revenue models.

**Accountability** – Who should be held responsible for the harm caused by software?

**Dependability** – Decision to maintain and/or keep a software product by software?

**Common goods** – Contributing to, using, promoting open source software.

**How can software engineers ensure they adhere to ethical standards in their work**

**Awareness and Proactive Thinking:** Before diving into development, think about how the software could be used and misused. Identify potential biases in algorithms, privacy concerns, and safety risks. Proactive consideration helps mitigate these issues early on.

**Prioritize user well-being**: Strive to create software that is fair, inclusive, and user-friendly. This might involve incorporating diverse perspectives in the design process and testing for bias in algorithms.

**Security and Privacy**: Build robust security measures to protect user data and prevent unauthorized access. Implement strong authentication and encryption protocols.

**Stay updated on evolving ethical considerations**: The technological landscape is constantly changing, and so are the ethical considerations. Attend workshops, read articles, and participate in discussions to stay informed about emerging ethical issues.

**Open communication and advocacy**: Voice your concerns if you see unethical practices being encouraged. Speak up about potential risks and advocate for ethical design principles within your team or organization.

**References**

-, M. I. H. (2023). Software Development Life Cycle (SDLC) Methodologies for Information Systems Project Management. *International Journal For Multidisciplinary Research*, *5*(5), 6223. https://doi.org/10.36948/ijfmr.2023.v05i05.6223

Božić, V. (2023). *VERSION CONTROL SYSTEM IN AGILE PROGRAMMING*. https://doi.org/10.13140/RG.2.2.23777.17769

Canfora, G., & Cimitile, A. (2001). *SOFTWARE MAINTENANCE* (pp. 91–120). https://doi.org/10.1142/9789812389718\_0005

Casteren, W. V. (2017). *The Waterfall Model and the Agile Methodologies: A comparison by project characteristics*. https://doi.org/10.13140/RG.2.2.36825.72805

Chakraborty, A., Baowaly, M. K., Arefin, A., & Bahar, A. N. (2012). *The Role of Requirement Engineering in Software Development Life Cycle*. *3*(5).

Dzerzhinskiy, F., & Raykov, L. D. (n.d.). *What Is Software Engineering?*

Freeman, J. (2022, June 2). Comparison Between Agile VS Waterfall. *Agile VS Waterfall: A Complete Comparison*. https://www.edrawsoft.com/agile-vs-waterfall.html

HEZAM, T. (2021). *Software project management*. 169591 Bytes. https://doi.org/10.6084/M9.FIGSHARE.14945535.V1

*IJSRD - International Journal for Scientific Research & Development| Vol. 6, Issue 12, 2019 | ISSN (online): 2321-0613*. (n.d.). *6*(12).

Iqbal, T. (2017). *Software Testing*. Unpublished. https://doi.org/10.13140/RG.2.2.18247.60323

Mall, R. (2015). *Fundamentals of software engineering* (Fourth edition, eastern economy edition). PHI Learning Private Limited.

Nuseibeh, B., & Easterbrook, S. (2000). Requirements engineering: A roadmap. *Proceedings of the Conference on The Future of Software Engineering*, 35–46. https://doi.org/10.1145/336512.336523

selleo.com. (2024, April 24). How Does Modular Software Architecture Improve Scalability? *How Does Modular Software Architecture Improve Scalability?* https://selleo.com/blog/how-does-modular-software-architecture-improve-scalability

Swanlund, S., Wilson, V., Kapoor, V., Gabbard, M., & Srinath, C. (2024). Software Development Life Cycle. *Journal of the Society for Clinical Data Management*, *4*(1). https://doi.org/10.47912/jscdm.343

Team, S. (n.d.). Software Design Principles [Blog]. *6 Software Design Principles Used by Successful Engineers*. Retrieved June 2, 2024, from https://swimm.io/learn/system-design/6-software-design-principles-used-by-successful-engineers#:~:text=Software%20design%20principles%20are%20general,maintainable%2C%20scalable%2C%20and%20efficient.

Thakkar, M. (2024, January 9). Challenges of Software Development [Blog]. *10 Software Development Challenges Every Developer Faces*. https://synoptek.com/insights/it-blogs/10-challenges-every-software-product-developer-faces/

thalesgroup.com. (n.d.). *What is Software Maintenance?* https://cpl.thalesgroup.com/software-monetization/four-types-of-software-maintenance#:~:text=Without%20maintenance%2C%20any%20software%20will,and%20essentially%20useless%20over%20time.&text=All%20software%20companies%20should%20have,important%20strategy%20in%20software%20development.

Umar, M. A. (2020). *Comprehensive study of software testing: Categories, levels, techniques, and types*. https://doi.org/10.36227/techrxiv.12578714.v2

Zolkifli, N. N., Ngah, A., & Deraman, A. (2018). Version Control System: A Review. *Procedia Computer Science*, *135*, 408–415. https://doi.org/10.1016/j.procs.2018.08.191