Introduction to Software Engineering

Zenas Kigen

Power Learning Project

Software engineering

July, 11, 2024

**Software Engineering**

Software engineering may be defined as the processes associated with creating and developing software. As a field of study, software engineering slightly differs from traditional programming, which may produce a sequence of statements that can help solve a particular problem. It represents cycles of identification, conceptualization, requirement analysis, design, coding and implementation, system integration and testing, deployment, and maintenance. Software engineering attaches great importance to sleight-of-hand methodology and tools for developing quality, dependable, reusable, and efficient software systems (Putte et al.,2023).

Software Development Life cycle is a model used in the development of software.

The Software Development Life Cycle, commonly referred to as SDLC, is a concept that outlines the sequence in which software is developed. This system describes many phases through which a software project passes before deployment and later during maintenance.

**Phases of Software Development Life Cycle of SDLC**

Planning and Requirements Engineering: It outlines its parameters, identifies users' needs, and specifies development workflows and tasks in detail.

Design: To be more precise, a system of components that can be divided into logical layers, as well as the components' interfaces and data structures, are described here.

Development: This phase entails translating, developing, and building the software system.

Testing: It is used in functional testing to verify that the software is functional and in performance testing to check how well it performs in speed and overall activity.

Deployment: This is for the end-users or directly for production use, and by the release of the software, more features of its use may be exposed.

Maintenance: After the software has been released to the market, it is constantly monitored and maintained, and new datasets are processed and enhanced for bug corrections.

The most suitable SDLC methodology could be chosen depending on the project's size and complexity and the workers' preferences and inclinations.

**Agile vs. Waterfall Models**

Waterfall Model: This is a linear and phase-oriented SDLC model in that each project phase is accomplished before the next phase is implemented (Tantascher & Mayer, 2022). It is a well-defined framework with the advantages of being structured and preplanned. Still, it has the drawbacks of rigidity and bureaucratic structure, which do not allow for adjustments to new specifications. The waterfall is ideal for implementation in projects with stable requirements that are challenging to alter in the progression of the development cycle.

Agile Model: Agile development is a process carried out in short cycles using the spiral approach, called iterations. It is deeply devoted to releasing solid first iterations in brief intervals called sprints while incorporating feedback. Agile is particularly useful for projects likely to have requirement changes in progress or if the end-users are critical to the final output.

**Requirements Engineering**

Requirements engineering refers to identifying, analyzing, documenting, and approving functional and non-functional needs for any software system. It helps to guarantee the technically optimal and suitable response to the requirements and expectations of the stakeholders (Kasauli et al., 2021). ALM is the strategic management of the application. It encapsulates everything related to it, starting from the identifiable requirements in the form of a well-defined set of requirements from which developers start their work and which eliminates costly rework at a later stage of the project.

**Software Design Principles**

Software design principles represent a system of behaviors engineers adhere to develop understandable, scalable, and easy-to-maintain software systems. Let us acquire various essential principles:

1) Modularity is a software development technique that divides a software system into smaller parts and reusable independent units. Every single module is unique; it also possesses a clearly stated goal and communicates with other modules via interfaces. This approach offers several benefits: This approach provides several benefits:

i. Improved Maintainability: The various components are not complicated, so one may alter, repair, or even uninstall an individual module without necessarily affecting the whole system.

ii. Enhanced Scalability: As for the new features, one can conclude that they can be added by creating new modules and extending them, thus placing the new functions under the existing framework of the program.

iii. Reduced Complexity: Modularity helps to comprehend, construct, and verify complex systems; these systems are more understandable and accessible to build and check as they are split into simpler parts.

**Testing in Software Engineering**

Testing is an essential step of software development that reassures software quality, reliability, and functionality (Khan et al.,2022). There are different levels of testing, each with a specific focus: There are different levels of testing, each with a particular focus:

i. Unit Testing: The testing process can be performed on specific single components of software, such as functions, classes, etc., to see whether they function in the manner they are intended to.

ii. Integration Testing: End-to-end tests involve testing each module simultaneously to check for compatibility and ensure that information passes from one module to another as required.

iii. System Testing: Integration testing: All software's functional and non-functional aspects are initiated to examine if they conform to expected standards.

iv. Acceptance Testing: Users and other interested parties then review the software to check whether it meets their specifications before application.

So, a single bug should be stressed and tested during development to correct it before the project is developed and fully warrants a recall.

**Version Control Systems**

Source management systems are also known as version control systems (VCS), which are programs that help keep track of changes made to the code and data. They allow developers to:

i. Maintain a history of changes: VCS tracks all the changes done on the codebase and can revert to a previous version for a specific code if needed.

ii. Facilitate collaboration: Async and parallel development is possible where several developers can work on a project simultaneously without causing conflict when merging.

iii. Improve code quality: Branching and merging in VCS allow or permit the developers to make several trials with conflicting code versions without disrupting the actual code.

Popular version control systems include:

i. Git: A distributed VCS you'll find employed extensively and recognized for flexibility branching and offline use.

ii. Subversion (SVN): It proposes a centralized version control model that is easier to use than Git but provides more limits.

**Software Project Management**

Software project management coordinates the institutions and resources needed to develop software to meet definite objectives on time and within cost estimates.

**Software Manager responsibilities**

i. Defining project scope and objectives: These encompass the specific features of the software as well as the projected time that the project shall take and the resources required to complete the same.

ii. Building and managing development teams: The project manager pulls together a suitably qualified and experienced team and creates a sense of teamwork.

iii. Risk management: Recognizing possibilities of adversity during the project's duration and following measures to control them.

iv. Communication and stakeholder management: The project manager reports to the stakeholders and responds to their concerns and expectations regarding the project—clients or sponsors.

v. Budget management: Monitoring the project's fiscal needs by controlling the costs of all materials, equipment, services, and workforce included in the project.

Competent software project management is essential in ensuring that books are developed and implemented on time and amicably within the set budgets and adhering to the set quality.

**Software Maintenance**

Software maintenance can be explained as the type of modification and evolution performed on a software application after its distribution to the users. There are three main types of maintenance activities:

i. Corrective Maintenance: This addresses the faults and inaccuracies that users provide and which are hindrances to its activities.

ii. Adaptive Maintenance: Customize software to fit the existing adjustments like new hardware, operating systems, or users.

iii. Perfective Maintenance: Modifications of software components' quality attributes that upgrade usability and quality to satisfy user needs and outcompete rivals.

Software maintenance thus holds a significant place in the software developmental cycle as software cannot remain static. Instead, it has to change over time to protect valuable data better.

**Ethical considerations and issues in Software Engineering**

i. Privacy: Admitting that protecting user data and its privacy and security is necessary. Software engineers should refrain from gathering and saving user data over the required data and provide the user data with correct security measures.

ii. Security: Comfort about not having software security assets becomes significant as it is now apparent that creating software resistant to harmful elements that could be used for evil purposes is vital.

iii. Bias: Similarly, algorithms and software systems can develop and replicate biases based on the data used to create them. Software engineers should also be aware of potential biases incorporated in their design software and try as much as possible to develop unbiased systems.

iv. Intellectual Property: Respecting others' rights concerning the fruits of their intellect is crucial. Software engineers should only steal source codes from others or other people's ideas if they are authorized.

Adherence to ethics helps develop and utilize the software for the appropriate impacts it can produce on society (Shneiderman, 2020). Software engineers can uphold ethical standards by Software engineers can uphold ethical standards by:

i. Recognizing that ethical codes exist in many professions in the course.

ii. Report possible ethical concerns about continuing specific products or services.

iii. Promoting effective software development management methodologies.

Identifying and acknowledging these ethical issues that prevail in software engineering can help engineers progress toward creating a more responsible technological environment (Sajid, 2024).

**Reference**

Putte, H. V., van der Voordt, T., & van Bortel, G. (2023). Performance measurement in the built environment

<https://pure.tudelft.nl/ws/files/173947748/www.linkedin.com>

Tantscher, D., & Mayer, B. (2022). Digital Retrofitting of legacy machines: A holistic procedure model for industrial companies. *CIRP Journal of Manufacturing Science and Technology*, *36*, 35-44.

<https://www.sciencedirect.com/science/article/abs/pii/S1755581721001760>

Sajid, A. (2024). Addressing Ethical Concerns: Guidelines for Quality Requirements in Requirements Engineering.

<https://aaltodoc.aalto.fi/items/a25591f9-a5ec-4ede-b918-dfe93cf4e6cd>

Shneiderman, B. (2020). Bridging the gap between ethics and practice: guidelines for reliable, safe, and trustworthy human-centered AI systems. *ACM Transactions on Interactive Intelligent Systems (TiiS)*, *10*(4), 1-31.

<https://dl.acm.org/doi/abs/10.1145/3419764>

Khan, R. A., Khan, S. U., Alzahrani, M., & Ilyas, M. (2022). Security assurance model of software development for global software development vendors. *Ieee Access*, *10*, 58458-58487.

<https://ieeexplore.ieee.org/abstract/document/9782440>

Kasauli, R., Knauss, E., Horkoff, J., Liebel, G., & de Oliveira Neto, F. G. (2021). Requirements engineering challenges and practices in large-scale agile system development. *Journal of Systems and Software*, *172*, 110851.

<https://www.sciencedirect.com/science/article/pii/S0164121220302417>