**Assignment 2-Introduction to Software Engineering**

Gideon Rotich

Power Learn Project

Introduction to Software Engineering: Assignment 2

June 11, 2024

**Define Software Engineering**

Software engineering is a well-organized field involved in creating, developing, and testing developing of software, maintaining a system of abstracts, methodologies, and tools to create high-quality software that meets stipulated criteria within the specified time and cost. Software engineering can be distinguished from programming mainly because it focuses on the rigorous engineering processes involved in developing software, including the requirement's specification, design, testing, and maintenance phase (Arvanitou et al., 2021).

**Software Development Life Cycle (SDLC)**

The Software Development Life Cycle (SDLC) is a framework that identifies stages of developing information systems. The typical phases of the SDLC are:

1. **Requirements Gathering and Analysis:** In this phase, the specifications of the to-be-built software are collected, documented, and evaluated to understand the current problem and the intended solution fully (Gurung et al., 2020).
2. **Design:** In the design phase, the structure or design of the software is created with the help of architecture, components, interfaces, and data structures, which are made according to requirements.
3. **Implementation of Coding:** This phase is where software codes are written according to the design specifications.
4. **Testing:** In this phase, the software is tested to discover problems that include defects and or flaws in the software. Testing can occur at different levels, including unit, integration, and system testing.
5. **Deployment:** After the software has been tested and recommended for use or if it is ready for usage, it is put into service or made available to the users or the production platform.
6. **Maintenance:** After the software has been deployed, there is always a provision to modify it to fix problems, eliminate bugs, and incorporate new needs that might not have been present earlier.

**Agile vs. Waterfall Models**

Agile and Waterfall are two strategies for managing organizations' software development process. The Waterfall model is a generalized and sequential software development model in which any phase (requirement, design, implementation, testing, deployment, or maintenance) cannot start until the previous phase is complete. This one is considered mechanical and highly bureaucratic as it centers on thorough prior preparation and documentation.

On the other hand, Agile is an iterative and incremental framework that thrives on flexibility, adaptability, and self-organization. It is a methodology that enhances the development process by separating it into segments with shorter periods of actual work and frequent examinations of the shortcomings that can be corrected in the upcoming work period. Scrum and Kanban are industry-standard process frameworks implementing full cross-functional collaboration and regularly delivering workable products.

However, it remains most appropriate where the requirements are clearly understood, and any scope or timeline changes are unexpected. The Agile model, however, is more suitable for undertaking projects that are recurrent or are changing in terms of their requirements since flexibility is critical.

**Requirements Engineering**

Requirements engineering is the initial step for building software systems, which involves acquiring, defining, documenting, and maintaining requirements for the system. It is widely described as appreciating the requirements in public, understanding the constraints, and explaining them by managing and establishing them to be precise and clear.

The requirements engineering process typically includes the following steps:

1. **Elicitation:** Getting requirements from diverse sources like expert and user interviews, online or physical questionnaires, and documentation analysis (Saeeda et al., 2020).
2. **Analysis:** Validate the requirements gathered and understand whether the gathered requirements provide a complete, consistent, and feasible picture of the venture.
3. **Documentation:** Representing the requirements properly so everything is transparent in its interpretation; this can be done with the help of use cases, user stories, or requirements specifications.
4. **Validation:** Make sure that the requirements outlined align with those of the stakeholders who share their expectations in writing.
5. **Management:** Controlling change and aging of the requirements over time in the lifecycle of the software.

Requirements engineering is essential for the same reason as it makes it possible for the development team to have a clear understanding of what the software system should do by way of addressing the needs and requirements of the stakeholders with the end in mind to avoid spending more cash in rework to fix developing a product that does not meet the expectations of the customer or even the failure of a project.

**Software Design Principles**

Modularity is one of the principles that have been advanced in software design. This principle involves distributing the system software in different sections that are self-contained, independent, and can be reused. Each module has clear interfacings, whereas they act autonomously and include particular functionality or responsibility.

Modularity improves the maintainability and scalability of software systems by:

1. **Reusability:** Due to the modularity of the design to which a given work is developed, the different components can be reused in other software sections or projects.
2. **Separation of Concerns:** Each module is dedicated to some particular worry or feature, which makes the code much more coherent and less complicated for further modifications.
3. **Testability:** The inherent assembly of structures, with several modules, enables the detection of a defect to be isolated in a simple procedure.
4. **Scalability:** Due to its modular design, it is possible to introduce new modules or alter existing ones as the system evolves without demonizing the overall system.
5. **Parallel Development:** One could develop a module while another team or a developer could work on another module simultaneously, thus improving productivity and decreasing the time spent on the development.

Componentization is, therefore, usually attained through object orientation, componentization, and service-oriented architecture.

**Testing in Software Engineering**

Testing can be concluded as an integral part of software engineering, which focuses on the system's quality assurance, reliability, and performance. Different levels of testing are performed throughout the software development lifecycle:

1. **Unit Testing:** This type involves testing and validating each software segment separately to ensure that each segment is correct and works as intended.
2. **Integration Testing:** It is a type of software integration where the components or modules of software are individually tested for integration to check the proper functioning when considered a complete system.
3. **System Testing:** This is system-level testing where the entire system is tested to check if it meets specified requirements and pinpoint any system-based issues.
4. **Acceptance Testing:** The final system is tested by the end-users or clients for functionality and suitability for the activity they will be used in or the projects they will be applied to.

Testing is crucial in software development because it:

* Identifying design defects and mistakes can help save time and money by rectifying them while the structure is still in its infancy
* Safeguards that the developed software has features that ensure it meets the required specifications and performs the intended task.
* It raises productivity and reliability and boosts the software's quality.
* The proposed product builds trust for the users by assuring the developers that it can handle various situations and conditions.
* It prevents faults and subsequent consequences, leading to financial losses or negative impacts on the organization's image.

**Version Control Systems**

Version control systems or VCS are applications used in the software development processes to control changes to the source code, documentation, and any other files associated with that particular project over a certain period. They allow for the throwing together of the code by different developers, tracking changes and documentation of change, resolution of conflicts and inclusion of the change, code review, and release, among other things (Wurzel Gonçalves et al., 2023).

Some popular version control systems include:

1. **Git:** An SCM that features branching, merging, and utilizing a shared repository for efficient development.
2. **Subversion (SVN):** A product-type system that resides at the center with only one source that programmers may check in and check out of code modification.
3. **Mercurial:** A version control software that resembles Git by being distributed software that is not bloated.

Authoring tools have several advantages, and the same applies to version control systems in the following ways:

* Sharing of deliverables and the ability to monitor changes made to the code or go back to a prior version when the need arises.
* Enabling everyone in a team to edit different parts of an application at once to, let’s say, work on some new feature or a bug simultaneously with others.
* It noted modifications to serve customer and developer debugging and auditing needs and comprehend the software's growth.
* Some of its advantages include facilitating code reviews and outlining quality assurance checks where team members can view changes proposed under the pull request before they are integrated into the mainframe.
* Supporting branching and merging such solutions allows for one or more development branches and integration of changes made to the code.

**Software Project Management**

The software project manager needs to have a range of skills and responsibilities, including, but not limited to, planning, scheduling, and updating the project's progress. Some key responsibilities and challenges faced by software project managers include:

**Responsibilities:**

1. **Planning and Scheduling:** The identification of specific objectives and targets for the project, the generation of requisite work plans, and the identification of necessary resources and their distribution throughout the project.
2. **Risk Management**: OCO stands for the procedure of identifying potential risks, evaluating risk scope, and setting possible measures (Albasteki, 2021).
3. **Resource Management:** Recruitment and selection of performers, preparation of staff for work on the project, and distribution of work among subordinates.
4. **Communication and Collaboration:** In this case, it involves enhancing communication and collaboration between team members and various stakeholders and assisting clients.
5. **Quality Assurance:** Quality assurance of the software and ensuring that it complies with the striking standards to the best quality level.
6. **Budget Management:** The real focus is those processes related to checking up and controlling the project's expenses, effective utilization of resources, and other issues connected with the financial side of the project.

**Challenges:**

1. **Scope Creep:** Of the three, the scope is visible in managing the project from conception to implementation.
2. **Resource Constraints:** Coordinating the work of separate sections and subdivisions and organizing their activity flows; personnel, material, and time resources are usually scarce (Berntzen et al., 2021).
3. **Team Management:** Personnel management includes assembling a solid and organized team with the required skills and characterized by diverse personality profiles.
4. **Risk Mitigation:** Risk management involves the assessment of threats that are likely to occur within the project and ensuring that they are prevented or minimized to acceptable levels.
5. **Stakeholder Management:** It has to do with the management of expectations and interaction with the public, customers, clients, and the end-users.
6. **Change Management:** Realizing the importance of accommodating development and integration features, client requirements, or project restrictions while causing minimal impact.

Software project management is a significant element in developing practical software projects that must be completed and delivered on time and with proper costs while maintaining team spirit.

**Software Maintenance**

Software maintenance is modifying, updating, and supporting a deployed and released system. Software maintenance is essential in the software life cycle because sometimes it is necessary to change:

* **To fix Defects:** This involves correcting errors, bugs, or faults that were not detected or solved during the preliminary development and testing phases.
* **Improving functionality** refers to adding new features, capabilities, or improvements to the software based on changing customer requirements or market trends (Nordal & El-Thalji, 2021).
* **To accommodate changes:** The aim is to modify the software to be compatible with altering operational environments such as new hardware, software platforms, or integration with other systems.
* **Optimizing performance:** This involves improving efficiency, speed, scalability, and software resource utilization due to changing workloads or usage patterns.
* **For regulation compliance:** This ensures software complies with updated legal, regulatory, or compliance requirements.

There are different types of maintenance activities:

* **Corrective Maintenance:** Rectifying defects/bugs in software
* **Adaptive Maintenance**: Modifying Software to Adapt to Changes in External Environments like New Hardware Platforms, etc.
* **Perfective Maintenance:** Improvement of the given software by increasing its value concerning its features, speed, or ease of use according to input from the user or other needs.
* **Preventive Maintenance:** Converting the program into another type of software for easier maintenance, increased reliability, or extendibility.

The software's maintenance process is essential to sustain the need for the software while in use, as well as its reliability and continued applicability throughout the lifecycle. The inability to perform maintenance on the software can lead to the point that the software becomes less efficient, gains more technical debt, and, in the long run, the system can develop faults or become obsolete.

**Ethical Considerations in Software Engineering**

Software engineers face numerous ethical challenges and considerations in their work, as their choices can profoundly affect individuals, organizations, and society. Some of the moral issues that might confront software engineers include:

* **Privacy and Data Protection:** Good care, storage, and safeguarding of sensitive users' data following privacy laws and best practices.
* **Intellectual Property Rights:** Adherence to intellectual property rights, no plagiarism or unauthorized use of any copyrighted material, and proper citation of sources.
* **Accessibility and Inclusivity:** Making software available to disabled individuals with disabilities and ensuring non-discrimination policies in software development.
* **Environmental Impact:** The environmental impact assessment of software development activities such as energy consumption, resource utilization, and e-waste generation.
* **Professional Integrity:** They must maintain professional integrity by not engaging in dishonesty, transparency, and accountability, avoiding conflicts of interest, and conforming to professional codes of conduct.
* **Safety and Security:** Developing strong security measures for software design to protect against vulnerabilities, cyber-attacks, or any possible harm to users or systems.
* **Ethical Use of Artificial Intelligence (AI):** Guaranteeing that AI systems meet the standards required for being FAIR with particular emphasis on how they are designed and used and whether or not they contain biases (Alzubaidi et al., 2023).

To address these ethical concerns, software engineers should strive to:

* Abide by various professional ethics and standards within the occupation/profession.
* Be a lifelong learner and start applying what the profession considers the most critical factor: knowledge updating concerning new ethics and regulations.
* Encourage open dialogue between stakeholders, including the clients, the users, and the community members.
* Always observe and apply the proper ethical standards and procedures at each phase of SDLC, including the requirement analysis phase and the deployment and maintenance phase.
* Voluntarily demonstrate a degree of openness in their behavior and actions and assume liability for the outcomes of their options and efforts.

Thus, software engineers must stick to the appropriate level of ethics and professionalism while retaining the outcomes that are useful for society and technically helpful in practice.

**References**

Albasteki, O. N. M. S. (2021). Corporate stakeholders, environmental and social risks, and enterprise risk management: towards an integrating framework (Doctoral dissertation, Brunel University London). <http://bura.brunel.ac.uk/handle/2438/23105>

Alzubaidi, L., Al-Sabaawi, A., Bai, J., Dukhan, A., Alkenani, A. H., Al-Asadi, A., ... & Gu, Y. (2023). Towards Risk‐Free Trustworthy Artificial Intelligence: Significance and Requirements. International Journal of Intelligent Systems, 2023(1), 4459198. <https://doi.org/10.1155/2023/4459198>

Arvanitou, E. M., Ampatzoglou, A., Chatzigeorgiou, A., & Carver, J. C. (2021). Software engineering practices for scientific software development: A systematic mapping study. Journal of Systems and Software, 172, 110848. <https://doi.org/10.1016/j.jss.2020.110848>

Berntzen, M., Stray, V., & Moe, N. B. (2021, June). Coordination strategies: managing inter-team coordination challenges in large-scale agile. In International Conference on Agile Software Development (pp. 140-156). Cham: Springer International Publishing. (pdf). <https://doi.org/10.1007/978-3-030-78098-2>

Gurung, G., Shah, R., & Jaiswal, D. P. (2020). Software development life cycle models-A comparative study. International Journal of Scientific Research in Computer Science, Engineering and Information Technology, 6(4), 30-37. : <https://doi.org/10.32628/CSEIT206410>

Nordal, H., & El‐Thalji, I. (2021). Modeling a predictive maintenance management architecture to meet industry 4.0 requirements: A case study. Systems Engineering, 24(1), 34-50. <https://doi.org/10.1002/sys.21565>

Saeeda, H., Dong, J., Wang, Y., & Abid, M. A. (2020). A proposed framework for improved software requirements elicitation process in SCRUM: Implementation by a real‐life Norway‐based IT project. Journal of Software: Evolution and Process, 32(7), e2247. <https://doi.org/10.1002/smr.2247>

Wurzel Gonçalves, P., Calikli, G., Serebrenik, A., & Bacchelli, A. (2023). Competencies for code review. Proceedings of the ACM on Human-Computer Interaction, 7(CSCW1), 1-33. <https://doi.org/10.1145/3579471>