International Academic Institute for Science and Technology

International Academic Journal of Innovative Research
Vol. 12, No. 4, 2025, pp. 33-37.

ISSN 2454-390X

International Academic Journal of Innovative Research

www.iaiest.com

Advanced Requirements Engineering and Use Case Analysis for Software Systems

Dr. Sanjay Kumar [a], Dr. Sapna Bawankar [b]

[a] *Assistant Professor, Kalinga University, Naya Raipur, Chhattisgarh, India.*
E-mail: ku.sanjaykumar@kalingauniversity.ac.in, Orcid: https://orcid.org/0009-0004-2958-2902
[b] *Assistant Professor, Kalinga University, Naya Raipur, Chhattisgarh, India.*
E-mail: ku.sapnabawankar@kalingauniversity.ac.in, Orcid: https://orcid.org/0009-0005-9651-582X

Abstract

The software design process encompasses activities and capabilities related to objectives, planning, implementation, and the modification of complex systems, all guided by the "requirements specification" before the actual programming begins. A software process model is chosen to guide the entire journey of software development, starting from the initial customer idea all the way to the final product release. In the realm of software engineering, we focus on four key phases, rather than trying to cover everything. These essential phases are analysis, design, implementation, and testing. They outline what needs to be built, how it will be constructed, deployed, and ultimately ensure it meets quality standards. Different process models adopt various analysis techniques, largely due to differences in frequency, application, and how the process is implemented. The software engineering methodology highlights these four crucial phases—analysis, design, implementation, and testing—which are vital for the software life cycle during product delivery. The results from these phases can also be applied to other stages of the software life cycle, including deployment, operations, maintenance, legacy support, and eventually, discontinuation, as systems evolve through multiple versions from start to finish.

Keywords: Software Engineering, Programming, Object-Oriented, Development.

DOI:10.71086/IAJIR/V12I4/IAJIR1232
Received: 18 September, 2025/Revised: 29 October 2025/Accepted: 18 November 2025/Published: 30 December 2025

Introduction

In essence, a software life cycle model outlines the proper way to develop software. According to Curtis et al. (1988), the descriptive model describes the development of a particular software system and can be used as a basis for comprehending and enhancing software development processes or for developing evidence-based prescriptive models. On the other hand, a prescriptive model provides guidance on how to develop a new software system. This model helps organize and structure the software development activities, guiding decisions and determining the sequence of tasks through established frameworks or guidelines (Heikkilä et al., 2015).

Software development projects often face a high failure rate, primarily due to unclear requirements and incomplete information, as highlighted by The Standish Group back in 1995. If the requirements aren't executed properly, it's likely that the projects will stumble, regardless of how well the subsequent phases are handled. This issue typically arises from a lack of clear requirements or poor communication between the analysis and design stages (Leffingwell & Widrig, 2000). By improving these two aspects of requirements engineering, we can significantly boost the overall success of software development projects. When requirements are incomplete or flawed, it leads to vague product definitions and inaccurate cost and schedule estimates, ultimately resulting in unexpected outcomes. If something isn't requested, it won't be produced, and conversely, what is requested may be delivered even if it's not truly necessary.

Motivation plays a crucial role in the productivity of software engineering professionals managing motivation remains a challenge and often feels undermined. Despite a growing recognition among professionals that motivation is a key issue, there hasn't been a comprehensive review of the literature that brings together the existing research on motivation within the software engineering field (Singh & Gill 2011).

Review of Literature

Software Engineers (SEs) as a distinct group of high achievers (Laplante & Kassab, 2022). Wynkoop and Walz (1998) suggest that Software Engineers differ from their non-software counterparts, noting "significant differences in personalities between IT employees and the general population." Conversely, Ferratt and Short (1986) challenge the notion of differences between IT and non-IT workers, finding that IT employees and managers are not necessarily more achievement-oriented than their non-IT peers, even though they identified that the work itself serves as a strong motivator for these IT groups.

In the realm of software, requirements engineers have started paying more attention to non-functional requirements, which Chung et al. (1999) refer to as global requirements. These non-functional requirements encompass quality attributes and constraints, playing a crucial role in software development. However, they tend to be analysed subjectively and can be challenging to evaluate. They can be synthesized by establishing a consensus around the classifications of these requirements. The communication between the system and its environment operates independently of how it's implemented (Issa & AlAli, 2011). Users and any other systems that communicate with it are part of the environment; they are known as functional requirements (Fig. 1).

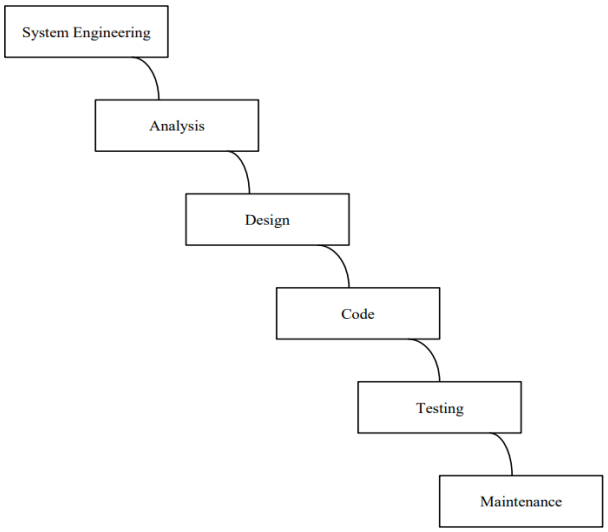


Fig. 1: Framework of the Study

Requirements engineering (RE) is frequently viewed as a beginning point in the field of software development. While that's generally true, it's important to note that requirements can shift during the development process and continue to evolve even after the system has been in use for a while. This makes RE a crucial element in managing changes throughout the software development lifecycle. However, most RE activities take place early on in a project's life. When requirements are flawed—whether due to misunderstandings or omissions—it can lead to costly fixes later on, as highlighted by (Bjarnason et al., 2016).

System Requirements

A well-organized paper contains the thorough explanations of the system services. Between the client and the contractor, this document serves as a contract. Software developers, client engineers, system architects, and system end users are among the target audience members. Essentially, this document outlines the functions that a system and its components are expected to perform, which we refer to as functional requirements (Armour & Miller, 2000). These requirements detail the inputs, behaviours, and outputs associated with each function. A functional requirement outlines the goals of a system, including computations, technological specifications, data processing, and other particular features. It's crucial for the system to define how it should respond to specific inputs and behave in various situations. The application of system architecture is guided by these functional requirements. Once a set of functional requirements has been gathered and validated, a requirements analyst will create use cases. Every use case uses one or more functional needs to demonstrate a behavioural scenario. Frequently, the analyst can obtain these functional requirements by beginning with a collection of UML use cases that need to be developed in order for users to carry out each use case (Fig. 2).

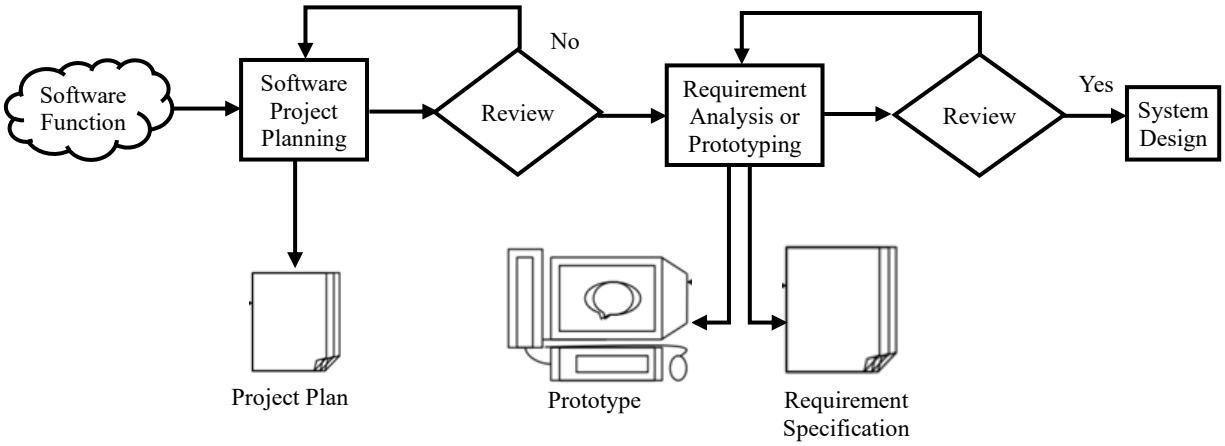


Fig. 2: Activities of Requirements Analysis

Derived requirements are essentially the needs that are implied within the broader set of Stakeholder requirements. It's important to ensure that we meet one or more of these derived requirements. Typically, these requirements evolve from higher-level specifications. While there are constraints and various considerations that come into play, they often aren't clearly outlined in the baseline requirements. The chosen architecture brings in various elements, including information assurance requirements and design aspects. Throughout the entire Systems Engineering Process (SEP), these derived requirements are shaped and refined through thorough requirements analysis.

Discussion

Here's the text we're looking at: Analysts have conducted a thorough study on the feasibility of the system and its functionalities, drawing from the available information. The primary aim of the organization is at the heart of this feasibility study. This analysis shows that the software product can indeed be realistically developed, considering factors like implementation, the project's contribution to the organization, budget limitations, and alignment with the organization's values and objectives (Fig. 3).

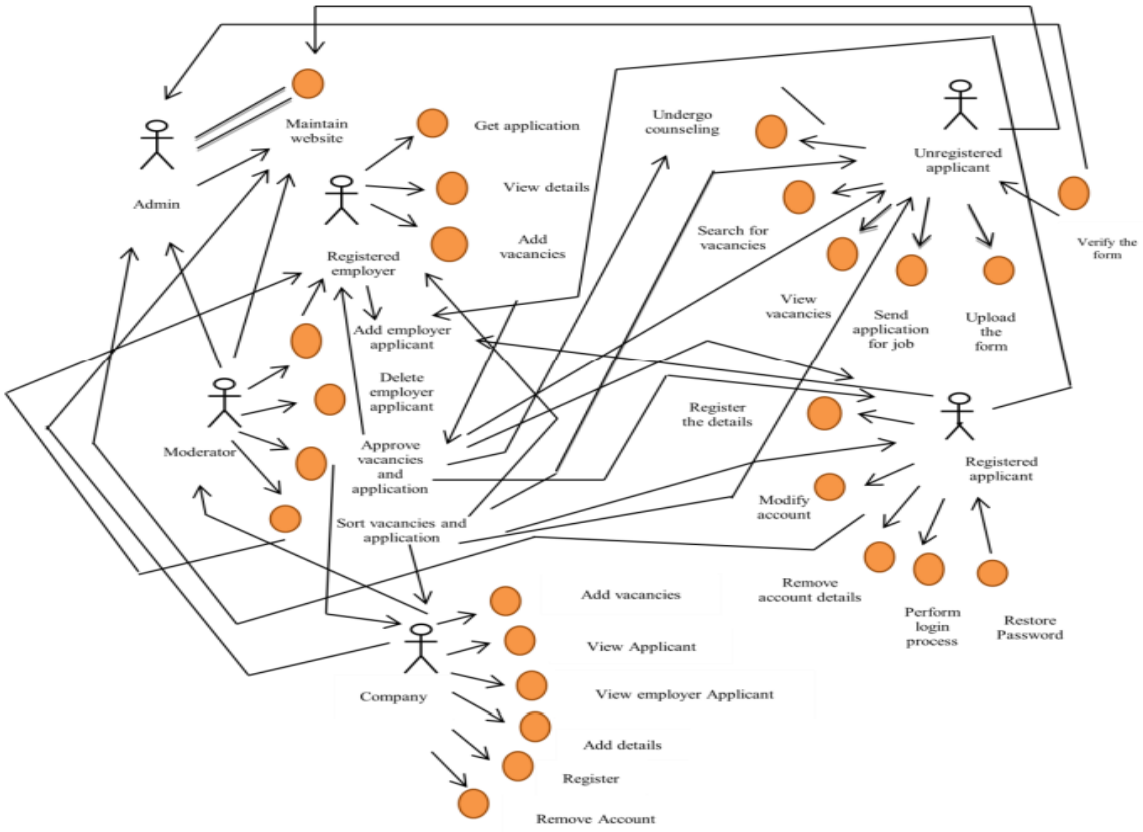


Fig. 3: Case Diagram of Employment Service

The project delves into various technical aspects like usability, reliability, maintainability, and productivity. The feasibility study report, which comes out of the feasibility study phase, should provide solid comments and recommendations for management on whether the project is worth pursuing. The main goal of requirements elicitation is to grasp the needs of users and project sponsors so that these needs can be effectively communicated to the system developers. A crucial part of this process is to draw out the needs of potential stakeholders (Jacobson, 1993). Initially, the idea is that gathering more requirements is always better than expected. During requirements elicitation, the information collected must be interpreted, analysed, modelled, and validated before requirement engineers can gather enough system requirements using various elicitation methodologies. Other activities in requirement engineering are closely tied to requirements elicitation. Information systems, especially large computer-based ones, handle and generate vast amounts of data (Sutcliffe et al., 1998). This data needs to be understood, manipulated, and managed effectively. Data modelling offers a way to tackle these challenges in requirements engineering. Entity Relationship Attribute (ERA) modelling has historically been the foundation of this type of modelling and analysis. However, ERA techniques are gradually being replaced by object-oriented modelling, which makes use of class and object hierarchies.

Requirements engineering isn't just about discovering and specifying requirements; it's also about fostering effective communication among the different stakeholders involved. The primary aim of requirements documentation is to ensure that it can be easily read, analysed, and validated.

Conclusion

Beyond merely examining particular behaviours, we also consider the non-functional needs when assessing a system's performance. These specifications are essential in determining a system's technical architecture. On the flip side, while the system design is guided by detailed functional requirements, the architecture itself is built upon these non-functional requirements. You can think of non-functional requirements as the quality benchmarks for a system. They encompass various quality aspects, including service quality goals. Every phase of the project lifecycle involves performance engineering, often known as systems performance engineering or software performance engineering. Roles, abilities, practices, activities, tools, and deliverables are all mixed together in this. Performance requirements—

like response time, throughput, latency, availability, and memory usage—are all part of the non-functional requirements that help shape the solution, ensuring it's designed, implemented, and supported effectively. In many companies, performance engineering has evolved into a distinct discipline that runs parallel to systems engineering, engaging multiple teams and individuals across the organization.

References

Heikkilä, V. T., Damian, D., Lassenius, C., & Paasivaara, M. (2015, August). A mapping study on requirements engineering in agile software development. In *2015 41st Euromicro conference on software engineering and advanced applications* (pp. 199-207). IEEE. https://doi.org/10.1109/SEAA.2015.70

Leffingwell, D., & Widrig, D. (2000). *Managing software requirements: a unified approach*. Addison-Wesley Professional.

Singh, N., & Gill, N. S. (2011). Aspect-oriented requirements engineering for advanced separation of concerns: A review. *International Journal of Computer Science Issues (IJCSI)*, *8*(5), 288.

Laplante, P. A., & Kassab, M. (2022). *Requirements engineering for software and systems*. Auerbach Publications. https://doi.org/10.1201/9781003129509

Issa, A. A., & AlAli, A. I. (2011). Automated requirements engineering: use case patterns-driven approach. *IET software*, *5*(3), 287-303. https://doi.org/10.1049/iet-sen.2010.0014

Bjarnason, E., Unterkalmsteiner, M., Borg, M., & Engström, E. (2016). A multi-case study of agile requirements engineering and the use of test cases as requirements. *Information and Software Technology, 77*, 61-79. https://doi.org/10.1016/j.infsof.2016.03.008

Armour, F., & Miller, G. (2000). *Advanced use case modeling: software systems*. Pearson Education.

Jacobson, I. (1993). *Object-oriented software engineering: a use case driven approach*. Pearson Education India.

Sutcliffe, A. G., Maiden, N. A., Minocha, S., & Manuel, D. (1998). Supporting scenario-based requirements engineering. *IEEE Transactions on software engineering*, *24*(12), 1072-1088. https://doi.org/10.1109/32.738340