ENGI 9838

Assignment 1

Author: Ziyang Zhou

Student#: 202193950

2024/1/28

Enhancing Requirements Engineering in Agile Software Development

Problem Identification

The core challenge in Agile software development lies in balancing flexibility with structured requirements management. The paper "Using Ontology to Enhance Requirement Engineering in Agile Software Process" [1] highlights persistent issues in Agile requirements engineering (RE), such as miscommunication between stakeholders and development teams. For instance, ambiguous terms like "user-friendly" in natural language descriptions often lead to divergent interpretations. Additionally, overreliance on informal verbal communication results in undocumented requirement changes, causing "historical requirement loss" during maintenance. Large-scale projects further amplify these issues, as lightweight Agile documentation struggles to align cross-team requirements. These problems stem from the inherent tension between Agile's "lightweight process" and the need for rigorous requirement traceability.

Existing Solutions

To address these challenges, the authors propose an ontology-based framework for structured RE. This framework leverages the OASys Methodology to transform textual requirements into hierarchical concept diagrams, visually mapping dependencies such as "payment functionality" and "security verification." Tools like OntoLT automate ontology extraction from unstructured text, generating XML-structured data that is then imported into Protégé for visualization. Experimental results demonstrate a 35% reduction in requirement clarification time and a 22% increase in stakeholder satisfaction. The framework's strength lies in preserving Agile's flexibility while introducing structured models to mitigate ambiguities, particularly in large-scale projects with frequent requirement changes.

My own Improvements and solutions

While effective, the ontology framework can be optimized for broader adoption. First, lightweight plugins like Slack could allow stakeholders to tag keywords in chat interfaces, automating ontology generation without requiring Protégé expertise. Second, coupling the framework with Git version control would enable dynamic requirement tracking—automatically highlighting changes such as removing nodes, and notifying relevant parties. Finally, creating reusable domain-specific ontology libraries would reduce redundancy, enabling teams to adapt templates rather than building models from scratch. From my perspective, these enhancements are particularly valuable for small teams with limited budgets, as they extending the framework's applicability beyond large projects.

Goal-Oriented Requirements Engineering for Complex Systems

Problem Identification

The paper "Goal-Oriented Requirements Engineering: From System Objectives to UML Models to Precise Software Specifications" [2] addresses a fundamental challenge in requirements engineering: managing conflicts and ambiguities in complex system objectives. Traditional approaches often struggle to systematically decompose high-level goals, such as ensure data security or optimize user experience, into actionable software specifications, especially when multiple stakeholders have competing priorities. For example, in a healthcare information system, the goal of maximizing patient privacy might conflict with enabling real-time data sharing among medical staff. The authors emphasize that manual goal decomposition heavily relies on human expertise, leading to inconsistencies, overlooked dependencies, and incomplete requirements—particularly in large-scale, multi-agent systems.

Existing Solutions

The study introduces the KAOS (Keep All Objectives Satisfied) methodology, a systematic framework for goal-oriented requirements engineering. This approach integrates multiple modeling techniques:

- AND/OR goal diagrams to hierarchically decompose system objectives, for example, breaking down "ensure data security" into sub-goals like "encrypt user data" and "restrict unauthorized access."
- UML use case and sequence diagrams to operationalize goals into software services, for example, mapping "optimize user experience" to specific use cases like "one-click login" or "personalized dashboard."
- Formal specification and obstacle analysis to detect conflicts and vulnerabilities, for example, identifying scenarios where "real-time data sharing" might inadvertently expose sensitive patient information.

The methodology was validated through case studies in information systems and process control domains, demonstrating improved requirement consistency and conflict resolution efficiency.

My own Improvements and solutions

While KAOS provides a robust theoretical foundation, its practical adoption faces barriers. First, integrating machine learning algorithms could automate conflict detection—for example, training models to flag contradictory goals during early requirement elicitation. Second, combining KAOS with Agile iterative feedback loops would enhance flexibility, allowing teams to refine goals incrementally based on stakeholder input, for example, adjusting privacy protocols after user testing. Finally, developing low-code visualization tools (e.g., drag-and-drop goal diagram editors) would democratize KAOS for non-expert teams, reducing reliance on formal specification expertise. These enhancements would bridge the gap between theoretical rigor and real-world usability.

Stakeholder Collaboration Intensity and Agile Team Performance

Problem Identification

In Agile development, everyone preaches "collaborate more with stakeholders," but reality often falls short. This paper, "The Connection of the Stakeholder Cooperation Intensity and Team Agility in Software Development,"[3] reveals that most teams focus only on obvious roles like customers and business units, while sidelining technical partners—IT support, compliance teams, or suppliers. For example, in a financial system project, teams might hold daily meetings with business departments but rarely involve legal or operations teams. Projects get stuck in late-stage compliance reviews or technical roadblocks, leading to endless requirement changes and missed deadlines. The study's real-world cases in financial firms show a counterintuitive twist: teams with higher agility scores thrived by working closely with engineers and suppliers, not by obsessing over customer feedback as traditional Agile principles suggest.

Existing Solutions

The authors propose a pragmatic fix: a four-layer stakeholder management framework (Environment, Offering, Product, Service Delivery). For instance, the Environment layer handles government regulations, while the Product layer focuses on developers and suppliers. They also designed a 1–10 scoring system to quantify collaboration intensity. In experiments with two European financial teams, Team B which is with higher agility scores maxed out collaboration scores with technical roles—9/10 with suppliers and a perfect 10 with internal engineering teams. Meanwhile, collaboration with end-users scored modestly, flipping traditional Agile wisdom on its head. This suggests that boosting team agility might rely more on "technical allies" than customer-centric approaches.

My own Improvements and solutions

While the framework work, it could be more practical. First, use machine learning to adjust collaboration priorities based on project phases—like prioritizing business stakeholders early and technical teams during testing. Second, integrate the scoring system with tools like Jira to auto-flag. For conflicting goals, like "fast delivery" and "strict compliance", implement quick voting systems for on-the-spot decisions. Finally, build industry-specific template libraries to skip repetitive setup work. I think these tweaks would make the framework accessible to small teams drowning in complex stakeholder dynamics.

References

[1]. S. Sitthithanasakul and N. Choosri, "Using ontology to enhance requirement engineering in agile software process," *2016 10th International Conference on Software, Knowledge, Information Management & Applications (SKIMA)*, Chengdu, China, 2016, pp. 181-186, doi: 10.1109/SKIMA.2016.7916218.

[2]. A. van Lamsweerde, "Goal-oriented requirements engineering: from system objectives to UML models to precise software specifications," *25th International Conference on Software Engineering, 2003. Proceedings.*, Portland, OR, USA, 2003, pp. 744-745, doi: 10.1109/ICSE.2003.1201266.

[3]. T. Juhola, M. H. Yip, S. Hyrynsalmi, T. Mäkilä and V. Leppänen, "The connection of the stakeholder cooperation intensity and team agility in software development," *2014 IEEE International Conference on Management of Innovation and Technology*, Singapore, 2014, pp. 199-204, doi: 10.1109/ICMIT.2014.6942425.