Report 1:

Software Development Life Cycle Early Phases and Quality Metrics: A Systematic Literature Review

Shokhista Ergasheva, Artem Kruglov, Innopolis University

Introduction

Software development methodologies, especially Agile, have reshaped the traditional Software Development Life Cycle (SDLC), which encompasses defining, developing, testing, delivering, operating, and maintaining software. Early defect detection plays a crucial role in reducing costs and improving team efficiency. This paper conducts a Systematic Literature Review (SLR) to classify early SDLC phases, explore software quality evaluation models, and identify metrics that enhance team and process performance, focusing on Requirements and Design phases.

Review Protocol

The SLR focused on key terms like "SDLC phases" and "quality metrics" in data sources such as Scopus, ResearchGate, and Web of Science. Over 200 publications were reviewed, with 60 primary studies retained after filtering for relevance. The review primarily assessed software quality evaluation in early SDLC phases, particularly Requirements Management and Design, which were discussed most frequently across studies.

Results

The review found that most studies focus on the early SDLC phases of Requirements Management and Design. The most common breakdown showed 31% of studies concentrated solely on Design, while 24% combined Requirements and Design. Key metrics used to evaluate software quality included Requirement Defect Density, Specification Change Requests, and Design metrics like Cyclomatic Complexity, which help assess reliability, maintainability, and functionality. Tools like CAME, ESQUT, and machine learning techniques were explored for automating quality assessments.

Discussions

The study confirmed that Requirements Management and Design are central to early SDLC activities, with 75.7% of studies emphasizing these phases. Various models, such as CAME tools and Source Monitor, were used to evaluate software quality in early phases. Fuzzy set theory was applied to handle uncertainty in metrics, while machine learning techniques were employed for more automated evaluations. Metrics like Chidamber and Kemerer's OO metrics, Cyclomatic Complexity, and Lines of Code were commonly used to measure software quality.

Conclusion

For a successful software project, consistent tracking and analysis of SDLC are essential, particularly in the early phases. The metrics used to evaluate software quality can vary by methodology and company goals. Future research should address uncertainty in software metrics and refine methods for software quality evaluation. As Benjamin Franklin wisely stated, "The bitterness of poor quality remains long after the sweetness of low price is forgotten." Ensuring quality early on is crucial for achieving a successful final product.

Report 2:

Analysis of SDLC Models for Embedded Systems

Sanket Dessai, K. Indu, V. S. Yerragudi, K. R. Narasimha Murthy, Mahita Patel, P. Padmapriya Dharishini, Suchit Kumar, N. D. Gangadhar

Introduction

Embedded systems play a pivotal role in modern applications, driven primarily by software. Modeling these systems based on their requirements is crucial to managing their complexity, especially in the development process. The Software Development Life Cycle (SDLC) provides a framework for managing software projects, and choosing the right SDLC model is critical for success. This paper analyzes various SDLC models and compares them with Agile Methods (AM) to assess their suitability for embedded system

development, which often requires specific attention to documentation and optimized code.

Agile Method

Agile software development emphasizes flexibility and collaboration, focusing on evolving requirements and solutions. Agile methods, such as Extreme Programming (XP), Scrum, and Feature Driven Development (FDD), promote rapid cycles, frequent iterations, and direct customer feedback. While Agile's flexibility makes it ideal for many software projects, embedded systems present challenges due to the need for optimized code and extensive documentation. Agile's continuous refactoring and iterative approach are beneficial for optimization in embedded systems, but the model needs to adapt to handle the unique documentation and design requirements of these systems.

SDLC Selection Factor

Choosing the appropriate SDLC model depends on the project's characteristics, such as system complexity, modularity, and resource availability. Waterfall, V-shaped, Spiral, Incremental, and Agile models each have their strengths. For complex systems, models like Waterfall and Spiral provide strong project management and documentation, while Agile offers flexibility and rapid iteration, albeit with some trade-offs in documentation. For embedded systems, where requirements are often unclear and technology may be unfamiliar, Agile can be effective if adjusted for the specific needs of the system.

Application of Agile Methods

Agile methods offer several benefits for embedded systems, including fast turnaround, better project management, and improved communication. However, challenges arise from Agile's emphasis on coding over documentation, making it difficult to meet the long-term maintenance needs of embedded systems. Additionally, while Agile's iterative approach aids in optimization, the system's inherent complexity often requires a more structured approach to design and documentation. Agile's test-driven development and continuous feedback mechanisms are advantageous, but debugging at the system level can be more complex due to hardware constraints.

Lean Agile Approach

The Lean Agile approach combines the principles of Lean, which focuses on minimizing waste, with Agile's flexibility to optimize embedded systems development. Lean Agile

emphasizes eliminating waste, empowering teams, and delivering fast, high-value features. By using user stories to structure development, teams can focus on high-priority features and improve overall efficiency. Lean Agile's fast turnaround, prioritized projects, and direct feedback loops make it highly effective for embedded systems, but it still requires adaptation to ensure sufficient documentation and design processes are maintained.

Conclusions

Agile methodologies offer significant advantages for embedded systems development, particularly in terms of flexibility, optimization, and customer interaction. However, the lack of emphasis on design and documentation in Agile methods poses risks, especially for long-lived embedded systems. To overcome these challenges, Agile methods must evolve to incorporate better design processes and ensure adequate documentation without sacrificing the benefits of agility. With these improvements, Agile can become a more robust approach for embedded system development.