<u>PHASE 1 DAY 29</u>

1. Early Stages of Software Development Life Cycle and Quality Metrics: A Simple Overview

<u>Introduction</u>
Modern software development methods, especially Agile, have changed how the Software Development Life Cycle (SDLC) works. The SDLC includes steps like planning, coding, testing, releasing, and maintaining software. Fixing problems early in the process helps save money and boost team efficiency. This review looks at the early stages of the SDLC—requirements and Design—to find ways to measure software quality and improve team performance.

<u>Research Methodology</u>
This study reviewed terms like "early SDLC stages" and "quality metrics" using Scopus, ResearchGate, and Web of Science databases. From 200 studies, 60 were chosen as the most relevant. The main goal was to learn about quality evaluation in the early SDLC phases, focusing on Requirements Management and Design.

<u>Key Findings</u>
Focus on Requirements and Design Phases
Most studies focus on the early SDLC stages. About 31% of the studies looked only at Design, while 24% covered both Requirements and Design.

<u>Common Quality Metrics</u>
Important metrics for measuring quality include:
Requirement Defect Density: Counts errors in requirement documents.
Specification Change Requests: Tracks how often requirements need updates.
Cyclomatic Complexity: Checks how complex the software design is to measure reliability and ease of maintenance.

<u>Tools and Techniques</u>
Some tools and technologies used for quality checks include:
CAME and ESQUT Tools: Automated tools for evaluating quality.
Machine Learning: Used to improve defect detection.
Source Monitor and Fuzzy Logic: Tools that handle uncertainty in metrics and automate quality checks.

<u>Discussions</u>
Requirements Management and Design are crucial for early SDLC success. About 75.7% of the studies focused on these stages. Tools like CAME and Source Monitor give insights into software quality early on. Fuzzy Logic helps manage unclear metrics, and machine learning speeds up evaluations. Metrics like Cyclomatic Complexity, Object-Oriented Metrics, and Lines of Code are reliable ways to measure software quality.

Conclusion
Checking and improving software quality early in the SDLC is critical for project success. Metrics and tools should match the company's goals and development methods. Future research should focus on improving ways to handle unclear metrics and making quality evaluation tools better. As Benjamin Franklin said, "The bitterness of poor quality remains long after the sweetness of low price is forgotten." Ensuring quality during the Requirements and Design stages is the key to successful software.

2. Analysis of SDLC Models for Embedded Systems

Introduction

Embedded systems are integral to modern applications and rely heavily on software. Proper SDLC modeling is crucial for handling their complexity. This paper examines different SDLC models and compares them with Agile methods to determine their suitability for embedded systems, which require thorough documentation and optimized code.

Agile in Embedded Systems

Agile promotes flexibility and frequent iterations through methods like XP and Scrum. While Agile's adaptability suits general software projects, embedded systems pose unique challenges like stringent documentation and hardware integration. Agile\u2019s iterative and refactoring methods help optimize embedded systems but require adjustments to meet design and documentation needs.

SDLC Selection for Embedded Systems

The choice of SDLC model depends on project complexity, modularity, and resources. \n- Waterfall and Spiral: Offer strong management and documentation for complex projects.\n- Agile: Provides flexibility and faster iterations but may lack adequate documentation for embedded systems.
Agile, if modified, can work for projects with uncertain requirements and evolving technologies.

Benefits and Challenges of Agile

Advantages:\n- Faster delivery.\n- Improved collaboration and feedback.\n- Optimized designs with iterative refinements.

Challenges :\n- Insufficient focus on documentation for long-term maintenance.\n- Debugging is harder due to hardware constraints.

Lean Agile for Embedded Systems

Combining Lean\u2019s waste reduction principles with Agile\u2019s flexibility creates an efficient development process. Lean Agile prioritizes user stories and high-value features, improving speed and efficiency. However, it still requires adaptation to ensure proper documentation and structured design.

Conclusion

Agile offers flexibility and customer-focused development for embedded systems. However, Agile must address its weaknesses in documentation and structured design to ensure long-term success. By evolving and incorporating these elements, Agile can become a more robust framework for embedded systems development.