Analysis of SDLC Models for Embedded Systems

1. Introduction

- **Software Development Life Cycle (SDLC)**: A framework that outlines activities in each stage of software development.
- SDLC Models include:
  - Waterfall Model
  - V-Model
  - Incremental/Iterative Model
  - Spiral Model
  - Agile Methods
- **Agile Methods** focus on flexibility, iterative development, and collaboration, offering a contrast to traditional models.

2. Agile Methods

- **Definition**: Agile is an iterative software development approach where requirements and solutions evolve through collaboration in self-organized teams.
- **Core Values** (Manifesto):
  - Individuals & interactions over processes/tools.
  - Working software over comprehensive documentation.
  - Customer collaboration over contract negotiation.
  - Responding to change over following a plan.
- **Popular Agile Frameworks**:
  - **Extreme Programming (XP)**: Focuses on coding cycles.
  - **Scrum**: Iterative approach with phases: Pre-game, Development, and Post-game.
  - **Crystal**: Scales methodology based on project size and criticality.
  - **Feature-Driven Development (FDD)**: Emphasizes the design and build phases.
  - **Dynamic Systems Development Method (DSDM)**: Fixes time/resources and adjusts functionality.
  - **Adaptive Software Development (ASD)**: Focuses on speculation, collaboration, and learning.

3. Selecting an SDLC Model

- **Factors**:
  - System type, size, complexity.
  - Resource availability, cost, and quality standards.
- **Model Suitability**:
  - **Waterfall/V-Model**: Suitable for well-defined requirements but rigid.
  - **Spiral/Iterative Models**: Handle unclear requirements, offer better visibility, and adaptability.

- **Agile**: Excels in dynamic environments but may lack documentation.

4. Application of Agile in Embedded Systems

- Benefits:
  - **Regular cycles**: Deliver executable features quickly.
  - **Continuous refactoring**: Optimizes code iteratively.
  - **Early testing**: Reduces bugs, aids debugging.
  - **Team communication**: Minimizes hardware/software integration issues.
- Challenges:
  - **Granularity**: Difficult to decompose embedded systems into small features.
  - **Documentation needs**: Embedded systems often require detailed records.
  - **Customer involvement**: Limited direct visibility in embedded projects.

5. Lean Agile Approach

- **Lean Principles**:
  - Eliminate waste (e.g., unused code, manpower).
  - Amplify learning through better team collaboration.
  - Deliver fast without compromising quality.
- **Benefits**:
  - Faster turnaround.
  - Direct feedback loop between teams and clients.
  - Prioritized features with high business value.
  - Efficient resource utilization.

6. Conclusion

- Agile methods improve flexibility, collaboration, and iterative delivery in software development.
- However, challenges like lack of documentation and integration difficulties exist in embedded systems.
- A hybrid approach like Lean Agile can address these issues effectively, ensuring optimized development processes.

Software Development Life Cycle Early Phases and Quality Metrics

1. Introduction

- **SDLC Overview**:
  - The Software Development Life Cycle (SDLC) consists of stages including defining, developing, testing, delivering, and maintaining software systems.
  - Early detection of defects in the **requirements** and **design phases** significantly reduces costs and improves software quality.
- **Objective of the Study**:
  - To classify early phases of the SDLC.
  - To identify quality metrics applicable to these phases.
  - To evaluate how early phase metrics impact software quality and team efficiency.

2. Research Methodology

- **Systematic Literature Review (SLR)**:
  - Search terms like *"Software development early life cycle phases and metrics"* were used across platforms like Scopus, ResearchGate, and Web of Science.
  - Over 200 publications were analyzed using inclusion and exclusion criteria to focus on studies relevant to SDLC early phases.
- **Research Questions**:
  1. How are SDLC phases classified?
  2. What models/approaches evaluate software quality in early phases?
  3. What activities are performed in these phases?
  4. What metrics are collected during these phases?

3. Results and Discussion

3.1. Classification of SDLC Phases

- Commonly identified early phases:
  - **Requirements Management Phase**: Involves activities like feasibility studies, requirements elicitation, analysis, and validation.
  - **Design Phase**: Establishes system architecture, design verification, and documentation.
- Alternative classifications (e.g., User Needs Analysis and Preliminary Design) emphasize activities like defining solution space and external behavior.

3.2. Approaches to Evaluate Software Quality

- **Quality Evaluation Tools**:
  - CAME (Computer Assisted Software Measurement and Evaluation)
  - ESQUT (for source code quality)
  - Source Monitor (for metrics like cyclomatic complexity)
- **Machine Learning Methods**:
  - Clustering techniques like fuzzy c-means and k-means for predictive analysis.
  - Metrics integration tools (e.g., Service Oriented Requirements Traceability Tool).
- **Key Metrics**:
  - Design metrics: Depth of Inheritance Tree, Cyclomatic Complexity.

- Documentation metrics: Completeness, readability, usability.

3.3. Activities in Early Phases

- **Requirements Phase**:
  - Activities: Feasibility study, elicitation, validation, and documentation.
  - Metrics: Requirement stability, defect density, traceability.
- **Design Phase**:
  - Activities: Architecture establishment, design review, and specification documentation.
  - Metrics: Cyclomatic complexity, cohesion, and coupling.

3.4. Metrics Collected

- **Requirement Metrics**:
  - Requirement defect density.
  - Requirement specification change requests.
  - Traceability of requirements.
- **Design Metrics**:
  - Cyclomatic complexity.
  - Maintenance severity.
  - Weighted Methods per Class (WMC).

4. Conclusions

- **Key Findings**:
  - Early SDLC phases focus heavily on **requirements** and **design**.
  - Metrics like defect density, design complexity, and traceability ensure process quality.
  - Tools and machine learning approaches enhance early-phase assessments.
- **Future Work**:
  - Explore methods to reduce uncertainty in metrics.
  - Develop more efficient, adaptable tools for early-phase evaluations.