# Research Methodology and Empirical Analysis

## Introduction

This document outlines the research methodology and empirical analysis for developing evaluation criteria for software artifact tracks and conducting an empirical study of repositories. The research consists of two main tracks: 1) Deriving comprehensive evaluation criteria with badges for software artifact tracks, and 2) Conducting an empirical analysis to evaluate the quality of software repositories using the derived criteria.

## Track 1: Deriving Evaluation Criteria for Software Artifacts

The first track focuses on developing a systematic methodology to derive a standardized set of evaluation criteria for software artifacts across different conferences. The aim is to analyze guidelines from multiple artifact tracks to create a universal set of evaluation standards.

### Research Questions

- How can a systematic methodology be developed to derive a standardized set of criteria for evaluating software artifacts across diverse conferences with artifact evaluation tracks?  
- What are the common themes, criteria, and best practices for README file content and repository organization as derived from the artifact guidelines of top software engineering conferences?  
- How do existing badge systems vary across conferences, and how can they be categorized to create a consistent evaluation framework?

### Steps and Techniques

1. \*\*Manual Review and Initial Thematic Analysis\*\*: Begin with a manual review of submission guidelines from 20-30 conferences to identify broad themes and common criteria.  
2. \*\*Organize and Label Textual Data\*\*: Extract relevant sections from each guideline, categorize them based on their content, and label them with initial themes for consistency.  
3. \*\*Topic Modeling to Uncover Themes\*\*: Use topic modeling techniques, such as LDA, to identify hidden themes across the guidelines. Tune the number of topics for meaningful results.  
4. \*\*NLP Techniques for Detailed Extraction\*\*: Utilize NLP to extract key entities and phrases related to evaluation criteria, followed by semantic similarity analysis to group related terms.  
5. \*\*Consolidate and Normalize Criteria\*\*: Combine related topics and phrases, standardize terminology across conferences, and develop overarching categories.  
6. \*\*Expert Review and Refinement\*\*: Conduct sessions with domain experts to validate and refine the criteria, ensuring their relevance and applicability.  
7. \*\*Assign Weights and Develop Scoring Rubrics\*\*: Create a weighted scoring system for each criterion, with a clear rubric for evaluation.  
8. \*\*Validation and Refinement\*\*: Test the criteria on sample repositories and adjust based on findings.

### Manual Review vs. NLP-Based Approach

For a dataset of 20-30 conference guidelines, manual review is a feasible approach. It provides a high-fidelity interpretation of nuanced technical standards, allowing for accurate and well-informed criteria development. While NLP techniques can be used to automate the process, they may overlook nuances and context-specific details critical to the evaluation process. Manual review ensures the depth and quality of the derived criteria.

## Track 2: Empirical Analysis Using Developed Criteria

The second track involves applying the derived evaluation criteria to a large set of software repositories. This empirical analysis aims to evaluate the quality of these repositories, validate the criteria, and explore how different guidelines affect artifact quality.

### Research Questions

- How effective are the derived criteria in evaluating and distinguishing high-quality software repositories?  
- What insights can be gained from applying the criteria to repositories across different programming languages, domains, and projects?  
- How can LLMs be fine-tuned to automate the assessment of software artifacts based on the developed criteria?

### Steps and Techniques

1. \*\*Data Collection and Preparation\*\*: Gather a diverse set of repositories (e.g., 2000) from top conferences with accepted artifacts.  
2. \*\*Annotated Training Data\*\*: Manually annotate a subset (200-300 repositories) based on the derived criteria to create high-quality training data.  
3. \*\*Structured Representation of Repositories\*\*: Convert each repository into a structured format, focusing on README content, file structure, and other relevant documentation elements.  
4. \*\*Fine-Tune LLM for Evaluation\*\*: Train a large language model (LLM) to understand and apply the criteria to repositories. Use structured representations as input and annotated evaluations as output.  
5. \*\*Automated Repository Evaluation Pipeline\*\*: Develop a system to parse new repositories, generate evaluation prompts, and aggregate LLM scores.  
6. \*\*Validation and Iteration\*\*: Compare LLM-based evaluations with manual assessments, refine the model, and improve consistency across evaluations.

### Justification for the Mixed Approach

Combining manual and automated evaluation ensures high-quality and scalable assessment. Manual annotation provides a gold standard for LLM fine-tuning, while automated evaluation using LLMs allows for scalability across large datasets. This hybrid approach balances accuracy with efficiency.

## Conclusion

The research aims to develop a standardized set of evaluation criteria for software artifacts and apply them to a diverse set of repositories. By combining manual review, topic modeling, and LLM-based evaluation, the methodology ensures comprehensive and scalable evaluation practices. This approach provides a robust framework for software artifact evaluation, advancing both theoretical understanding and practical application in the software engineering community.