**Section 1: Objectives, Questions, and Metrics (GQM Approach)**

* **Objective**: Assess the maintainability of Java software projects, focusing on class understandability and size.
* **Questions**:
  1. How do class size and complexity impact understandability in software projects?
  2. Do larger classes or those with deeper inheritance negatively influence maintainability and understandability?
* **Metrics**:
  1. **Weighted Methods per Class (WMC)** for understanding class complexity.
  2. **Depth of Inheritance Tree (DIT)** to assess hierarchy depth.
  3. **Lines of Code (LoC)** as a measure of class size.

**Section 2: Subject Programs (Dataset)**

* **Criteria for Program Selection**:
  1. **Project Age**: At least 3 years old to ensure projects have undergone maintenance cycles, which typically include refactoring and updates.
  2. **Project Size**: At least 10,000 lines of code (LoC) to focus on projects of significant size, which are likely to have complex structures.
  3. **Contributors**: At least 3 contributors to reflect collaborative development and diverse code contributions.
* **Rationale**: Projects meeting these criteria likely have experienced multiple development phases and varying maintenance tasks. Larger, older projects with multiple developers provide a realistic context for analyzing maintainability.
* **Project Descriptions**:
  1. In this section, list each selected project, describing its purpose and primary functionality in 2-3 sentences.
  2. **Example Table for Project Attributes**:

| **Project Name** | **LoC** | **Age (Years)** | **Contributors** | **Description** |
| --- | --- | --- | --- | --- |
| Project A | 12,000 | 4 | 5 | Library management system |
| Project B | 15,000 | 3 | 6 | E-commerce backend API |
| Project C | 20,000 | 5 | 8 | Real-time chat application |
| Project D | 10,500 | 3 | 3 | Task management tool |
| Project E | 11,500 | 4 | 7 | Data visualization library |

**Section 3: Tool Description**

* **CK Metric Tool**: Describe the tool briefly, including the link to the GitHub repository and a citation if available. Mention that the tool uses static analysis to extract CK metrics, enabling you to measure WMC, DIT, and LoC for each class in the selected Java projects.

**Section 4: Results and Analysis**

* **Data Visualization**:
  + Use bar charts or line graphs to represent WMC, DIT, and LoC values for each project.
  + Identify trends and outliers within each project, pointing out classes with unusually high or low values.
* **Example Analysis**:
  + **Trend Analysis**: Discuss whether classes with higher WMC and DIT metrics show increased complexity, potentially affecting understandability.
  + **Outlier Identification**: Highlight any classes with extreme values (e.g., very high WMC or DIT), indicating areas that may hinder maintainability.

**Section 5: Conclusions**

* **Size and Maintainability**:
  + Based on the data, discuss the effect of size (LoC) on maintainability for each project. For example, larger classes may require more effort to understand, especially if they exhibit high WMC or DIT values.
  + Reflect on how maintainability is impacted by high complexity (WMC) and deeper inheritance hierarchies (DIT), potentially suggesting these as areas for refactoring.
  + Summarize any correlations found between LoC, WMC, DIT, and understandability across projects.

**References**

* List any papers, articles, or tools used, formatted per your academic requirements.

**Submission Instructions:**

* **GitHub Repository**: Upload the final report and raw data (C&K metric values for each project) to a GitHub repository. The raw data should ideally be organized in a readable format (CSV or JSON).
* **Link Submission**: Submit the GitHub link via Blackboard.