**1. Project Outline:**

* **Objective:** Detect architecture smells and monitor the evolution of design rules in software architecture over time.
* **Deliverables:**
  + A report or paper detailing the approach, findings, and results.
  + A prototype/tool for detecting architecture smells and design rule violations.
  + A case study or application on an open-source or enterprise system.

**2. Potential Steps:**

**Step 1: Background Research**

* **Goal:** Understand the theory behind architecture smells and design rule spaces.
* **Key Topics to Explore:**
  + **Architecture Smells**: Research common architecture smells such as cyclic dependencies, improper modularization, god components, high coupling, and inappropriate use of layers.
  + **Design Rule Spaces**: Explore what design rules are (e.g., high-level constraints on modules, layers, or interactions) and how these evolve in response to business or technical changes.
* **Relevant Literature:**
  + Books: *"Clean Architecture"* by Robert C. Martin, *"Software Architecture in Practice"* by Len Bass, or *"Design It!"* by Michael Keeling.
  + Academic Papers: Look for papers on architecture smells, design rule spaces, and software evolution.
* **Result:** A literature review section or summary explaining these concepts.

**Step 2: Define Architecture Smells & Design Rules for Your System**

* **Goal:** Formalize which architecture smells and design rules you will be tracking in the project.
* **Tasks:**
  + Choose a system architecture to analyze. You can use a pre-built open-source system like [Spring PetClinic](https://github.com/spring-projects/spring-petclinic) or any other open-source projects with well-defined architecture.
  + Identify key smells such as:
    - **Layer violations** (violating architecture layering principles).
    - **Cyclic dependencies** between modules.
    - **Component centralization** (a single component taking on too many responsibilities).
  + Define the **design rules** (e.g., modules should follow certain dependencies, interfaces should be decoupled from implementation, etc.).
* **Deliverable:** A specification document listing the architecture smells and design rules to be detected.

**Step 3: Detection Techniques for Architecture Smells**

* **Goal:** Build or use tools to detect architecture smells automatically.
* **Tasks:**
  + Identify tools and frameworks that can help you automatically detect architecture smells:
    - **Static analysis tools**:
      * **SonarQube**: Extensible open-source platform for static code analysis that can help identify code smells and violations.
      * **Lattix**: Offers dependency analysis, helping identify cyclic dependencies and other modularization issues.
      * **Arcan**: A tool designed specifically to detect architecture smells.
    - **Dynamic analysis tools** (for runtime architectural violations).
  + Integrate these tools into your codebase and run detection processes. Collect the data over time (or analyze historical versions of the software).
  + If tools are unavailable, build custom scripts (e.g., in Python or Java) that scan code repositories for known architectural smells.
* **Deliverable:** A functioning prototype for architecture smell detection stemming from an augmentation of available baselines (e.g., the tool Arcan).

**Step 4: Track and Analyze Evolution of Design Rules**

* **Goal:** Detect changes in design rules as the software evolves.
* **Tasks:**
  + Implement a mechanism to monitor architecture changes over time (e.g., by analyzing git logs, commit histories, or comparing code snapshots).
  + Use version control history to detect when a module or layer violates an existing design rule.
  + Build a visual representation of the system’s architecture evolution and how design rule violations emerge over time.
  + Conduct analysis to detect "drifts" or deviations from original design rules (e.g., a module that starts taking too many responsibilities over time).
* **Deliverable:** A report or dashboard showing the evolution of design rule compliance over time.

**Step 5: Case Study / Application**

* **Goal:** Apply your approach to a real-world or open-source system.
* **Tasks:**
  + Choose an open-source project or a pre-existing enterprise system with multiple versions available for analysis.
  + Apply your architecture smell detection and design rule evolution monitoring tools to the system.
  + Summarize the findings: What smells were detected? How did the design rules evolve? Were there any patterns in how the architecture degraded or improved over time?
* **Deliverable:** A detailed case study showing the effectiveness of your approach.

**Step 6: Synthesis & Reporting**

* **Goal:** Analyze findings and propose actionable improvements.
* **Tasks:**
  + Summarize findings in terms of architecture smells and design rule violations.
  + Propose solutions or design refactorings to eliminate detected smells.
  + Offer insights into how design rules should be defined to avoid architecture smells in future systems.
* **Deliverable:** A final report or academic paper summarizing the research, methodology, and outcomes.

**3. Research Approaches:**

**Exploratory Research:**

* Since architecture smells and design rule spaces are still an evolving area of study, you could take an exploratory research approach, trying different combinations of tools and techniques to detect these issues.
* Focus on gaps in existing tools and try to build custom heuristics or algorithms.

**Empirical Research:**

* Conduct experiments on multiple versions of a system to empirically evaluate the detection of smells and rule violations over time. Record results and evaluate how the architecture quality changes.
* Compare systems of different sizes or complexity to see if certain types of smells are more common in larger systems.

**Comparison Study:**

* You can compare different existing tools for detecting architecture smells (e.g., SonarQube vs. Lattix) and their effectiveness in detecting smells in real-world projects.
* Perform a comparative analysis on the evolution of design rules in different systems.

**Survey-Based Research:**

* Collect data from developers or architects in the industry to understand the impact of architecture smells and design rule violations in real-world systems.

**4. Tools & Frameworks**

**Architecture Smell Detection Tools:**

* **SonarQube:** Widely used for static analysis, it identifies code smells and can be extended to architecture analysis.
* **Arcan:** A tool specifically designed to detect architectural smells, particularly cyclic dependencies.
* **Lattix:** Helps in dependency structure analysis and can be used to identify architectural issues like cyclic dependencies and improper layering.
* **Structure101**: A tool to analyze the structure and modularity of codebases.

**Design Rule Evolution Monitoring:**

* **Git:** Using version control histories to track changes in the codebase and identify when design rules were violated or altered.
* **Architecture Decision Records (ADR):** Use ADRs to track design decisions and see how they evolve or are violated over time.
* **Diff Tools**: Use diff tools or scripts to compare system architecture snapshots across different software versions.

**Visualization Tools:**

* **Graphviz**: For visualizing the system's architecture as a graph, helping in detecting architecture smells like cyclic dependencies.
* **D3.js**: For building interactive dashboards to visualize architecture evolution and design rule compliance.

**5. Evaluation Metrics:**

* **Smell Detection Rate**: The number of architecture smells detected over time.
* **Evolution of Design Rule Violations**: Number of design rule violations as the software evolves.
* **Architectural Complexity**: Track the complexity (e.g., in terms of coupling or number of modules) to correlate with detected smells.
* **Refactoring Impact**: Evaluate how removing architecture smells or aligning with design rules impacts system quality metrics like maintainability or performance.