The project titled **"Software Community Smells Evolution Investigation at Scale"** focuses on exploring and analyzing **community smells** within large-scale software projects. Community smells refer to social and organizational issues that arise in collaborative software development environments. These include communication breakdowns, team structure issues, and misalignment between developers, which can negatively affect software quality, productivity, and sustainability.

Investigating the evolution of these community smells at scale implies looking into their progression over time in large, open-source or enterprise-level projects, and how they impact project outcomes such as code quality, bug resolution, and team productivity.

**1. Project Overview:**

* **Objective:** To investigate how software community smells evolve over time in large-scale projects, understand their impact on project health, and propose strategies for mitigating their effects.
* **Deliverables:**
  + A framework or methodology for detecting and analyzing community smells.
  + A comprehensive report documenting the evolution of community smells and their impact on software quality and team productivity.
  + Case studies on large-scale open-source projects.

**2. Key Concepts:**

**Community Smells:**

* **Definition:** Community smells are socio-technical issues in the development process that arise from poor communication, misaligned roles, or unbalanced workloads among team members. They are the "social equivalent" of code smells in collaborative software development.
* **Examples of Community Smells:**
  + **Organizational Silo:** When developers from different parts of the organization fail to collaborate effectively, leading to poor knowledge sharing.
  + **Black Clouds:** Centralized communication among a few developers who control much of the interaction, which can lead to bottlenecks.
  + **Truck Factor:** The risk of a project being dependent on a small number of core contributors, leading to potential problems if they leave the project.
  + **Code Ownership Issues:** Conflict or mismanagement regarding ownership of code areas, leading to fragmented responsibility and code degradation.

**Evolution Investigation:**

* The study of how community smells emerge, persist, or change over time as a project evolves.
* This includes identifying periods of heightened community smells, their correlation with project milestones (e.g., major releases), and their potential relationship to project outcomes like defect density, feature delivery time, or community attrition.

**Scale:**

* Investigating community smells **at scale** involves analyzing large datasets, possibly across multiple large open-source or enterprise-level projects, with diverse and geographically distributed developer communities.
* Scaling the analysis means leveraging automated data collection and analysis techniques to evaluate community behavior over time.

**3. Potential Steps:**

**Step 1: Research and Define Community Smells**

* **Goal:** Understand community smells in software development, their causes, and their effects on project quality.
* **Tasks:**
  + Conduct a literature review on community smells and their classification (e.g., *"Organizational Silo," "Black Clouds," "Truck Factor"*).
  + Study socio-technical relationships in software development (developer collaboration, communication patterns, etc.).
  + Look into how tools like GitHub, GitLab, or JIRA influence community behavior.
* **Deliverable:** A detailed summary of community smells and their relevance to software engineering.

**Step 2: Data Collection from Large Software Projects**

* **Goal:** Gather data from open-source or enterprise projects to track community behavior and identify community smells.
* **Tasks:**
  + Select open-source repositories with a large, active developer community (e.g., **Apache**, **Linux Kernel**, **Kubernetes**).
  + Extract data such as:
    - **Commit Histories**: To study how team members interact and contribute.
    - **Issue Tracking**: Analyze issue assignment, comments, and resolution patterns.
    - **Pull Requests**: Evaluate code review discussions, frequency of contributions, and cross-team collaboration.
    - **Communication Channels**: Study activity on mailing lists, Slack channels, or forums (if available).
  + **Tools for Data Collection**:
    - **GHTorrent** or **GitHub API** for extracting repository data.
    - **GrimoireLab** for analyzing developer activities across multiple collaboration tools.
* **Deliverable:** A dataset of community interactions from selected repositories over time.

**Step 3: Detection of Community Smells**

* **Goal:** Develop or adopt algorithms to detect community smells in large-scale software projects.
* **Tasks:**
  + Define metrics or heuristics to detect various community smells. For example:
    - **Organizational Silo**: Detect through the analysis of collaboration networks (e.g., low cross-team interactions or contributions).
    - **Truck Factor**: Measure the proportion of the project that depends on a small group of contributors.
    - **Black Cloud**: Identify developers who are central in communication patterns.
    - **Ownership Issues**: Detect conflicting contributions or areas of the codebase that lack clear ownership.
  + Use **social network analysis (SNA)** or **graph theory** to model and analyze developer interactions and contributions.
  + Implement tools or scripts to automate the detection of these smells at scale.
* **Deliverable:** An automated tool or framework for detecting community smells from repository data.

**Step 4: Evolution Analysis of Community Smells**

* **Goal:** Analyze how community smells evolve over time and correlate with key project events or outcomes.
* **Tasks:**
  + Track the emergence and resolution of community smells over time.
  + Correlate the presence of community smells with project outcomes, such as:
    - **Bug density**: Do projects with more community smells have higher bug rates?
    - **Code quality**: Does the presence of ownership issues affect code maintainability?
    - **Release delays**: Do organizational silos or communication bottlenecks lead to slower feature delivery?
  + Use visualization tools like **Gephi**, **D3.js**, or **Graphviz** to represent community smells evolution.
* **Deliverable:** A comprehensive analysis of the evolution of community smells and their impact on the health of software projects.

**Step 5: Mitigation Strategies and Best Practices**

* **Goal:** Propose strategies to mitigate community smells based on the analysis.
* **Tasks:**
  + Based on the findings, suggest best practices for improving communication and collaboration in large-scale software projects.
  + Propose **tooling improvements** (e.g., enhancing issue assignment algorithms, better team communication strategies).
  + Study the impact of refactoring project teams or adjusting workflows to reduce the risk of community smells.
* **Deliverable:** A set of recommendations or best practices to help project managers and teams mitigate community smells.

**4. Research Approaches:**

**Empirical Research:**

* Conduct empirical research by gathering large datasets from multiple open-source projects. Measure the occurrence of community smells over time and evaluate the relationship between community smells and software quality indicators (e.g., defect rates, delivery time).

**Qualitative Research:**

* Interview or survey key contributors in open-source projects to gain insights into how they perceive community dynamics and the impact of certain behaviors on software quality and collaboration.

**Case Study Approach:**

* Perform case studies on a few well-known projects (e.g., **Kubernetes**, **Mozilla**, **OpenStack**) to track the evolution of community smells over time. Provide detailed insights into how community smells emerged, their effects, and the outcomes of mitigation strategies.

**5. Tools & Frameworks:**

**Data Collection & Analysis:**

* **GHTorrent:** A service that provides offline data related to GitHub activities (commits, issues, pull requests).
* **GrimoireLab**: An open-source platform for software development analytics, providing data from Git repositories, mailing lists, issue trackers, and more.
* **GitHub API**: For extracting commit, issue, and pull request data from GitHub repositories.

**Social Network Analysis Tools:**

* **Gephi**: A powerful tool for network analysis, allowing you to visualize and study the relationships between developers and teams.
* **D3.js**: A JavaScript library for producing dynamic, interactive data visualizations in web browsers.
* **Graphviz**: A graph visualization software used to represent developer collaborations.

**Community Smell Detection Tools:**

* **KAIAULU:** A tool specifically designed for detecting community smells in software projects by analyzing collaboration patterns in version control systems.

**6. Evaluation Metrics:**

* **Community Smell Detection Rate**: Number of community smells detected over time.
* **Code Quality Metrics**: Compare projects with and without community smells in terms of defect density, maintainability, and technical debt.
* **Developer Productivity**: Measure the impact of community smells on task completion time, issue resolution speed, and developer engagement.
* **Project Health**: Analyze how the presence of community smells correlates with release delays, bug frequency, or contributor churn.