**🎯 Your Project Objectives**

By the time you finish this project (end of thesis timeline), your objectives would look like this:

1. **Understand & Extend SonarQube**
   * Learn how SonarQube’s static detection & prioritization works.
   * Hook into SonarQube APIs or result feeds (not rewriting its engine, but building *on top*).
2. **Develop ML/AI Enhancements**
   * Train ML models to reduce false positives (Detection Engine domain).
   * Create a feedback-driven prioritization system that *learns from developer/team behavior* (Prioritization + Behavioral domains).
   * Integrate project history (commits, bug reports, past fixes) to re-rank/prioritize smells dynamically (Defect Density Overlay + Prioritization).
3. **Automate Ticket Creation & Ranking**
   * Automatically push prioritized smells into Agile tools (e.g., Jira, GitHub Issues) as actionable tickets.
   * Rank/re-rank tickets dynamically as new developer feedback or bug history comes in.
4. **Empirical Evaluation**
   * Measure whether your enhancements reduce false positives, improve prioritization accuracy, and help developers fix the *right* problems faster.
   * Publish metrics: precision/recall of smell detection, developer validation rates, ticket resolution rates, etc.

## **🏆 What You Gain at the End**

By completing this project, you will have:

* **A research contribution** → Novel ML/AI integration with SonarQube for prioritization & feedback loops.
* **A working prototype tool** → A system that takes SonarQube results → re-prioritizes with ML → pushes to Jira/GitHub.
* **Stronger research profile** → Your thesis + possible publications (software engineering, AI/ML for SE).
* **Practical DevOps skillset** → SonarQube, ML pipelines, Jira/GitHub API integration, Agile workflows.
* **Clear career positioning** → Industry-ready in software quality automation, AI-driven DevOps, or research in SE/AI

🔗 **How Your Topic Maps to SonarQube’s 5 Domains**  
Here’s the **exact tie-in** between your topic *“Automated Code-Smell Prioritization Integrated with Agile Platforms for Automatic Ticket Creation and Ranking”* and the SonarQube weaknesses:

**1. Detection Engine (Code Smell Search)**

* Weakness you tackle: **False positives** in smell detection.
* Your angle: Train an ML model (maybe lightweight classifiers or embeddings) that learns from project history + developer feedback which “smells” are *actually actionable*.

**2. Prioritization Logic**

* Weaknesses you tackle:
  + Static severity levels.
  + No historical context.
* Your angle: Build a **re-prioritization engine**:
  + Use project history (bugs, commits, frequency of changes) to **adjust severity dynamically**.
  + Use developer feedback loop to **rank what matters most now**.
* ✅ This is the **heart of your thesis**.

**3. Code Health Metric *(optional, merged into #2)***

* Weakness: Misleading aggregate score.
* Your angle: Instead of a raw count, **compute a dynamic health score** weighted by your ML prioritization.
* Could be a nice side contribution, but don’t let it derail you.

**4. Defect Density Overlay**

* Weakness: Ignores historical bug fix history.
* Your angle: Bring in Jira/GitHub bug history → train your prioritization engine with real “past defect hotspots”.
* This connects naturally with your “automatic ticket creation” since you already touch Jira APIs.

**5. Behavioral & ML Patterns**

* Weaknesses you tackle:
  + No feedback loop.
  + No personalization per project.
* Your angle: **Feedback Loop ML System** → Developers re-rank tickets or mark false positives → model retrains.
* This is your **novel AI research piece**: turning SonarQube static rules → into a living adaptive detection system.

Covering **5 weaknesses across 4 domains** (Detection, Prioritization, Defect Density, Behavioral Patterns).