Task 2A — Design Smells & Static Analysis Report

1. Tools Used

The system was analyzed using:

- **SonarQube** – for code-level smell detection, complexity, duplication, and maintainability issues.

- **DesigniteJava** – for detection of architectural and design smells.

This combination enables both **fine-grained code analysis** and **system-level design analysis**.

2. SonarQube Analysis

The entire project repository was analyzed using SonarQube. Issue data was exported via SonarQube Web API in JSON format (`~2300 issues`).

2.1 Overall Code Quality Metrics

Metric	Value
Lines of Code (LOC)	~67,000
Total Code Smells	~2,300
Duplication	3.8%
Reliability Issues	246
Security Issues	122
Test Coverage	0%

These numbers indicate a **large system** with **significant maintainability debt**.

2.2 Nature of Code Smells (from JSON analysis)

The exported JSON files revealed recurring categories:

Pattern Observed	Sonar Rules	Interpretation
High cognitive complexity	`S3776`	Complex “brain” methods
Generic exception usage	`S112`	Poor error abstraction
Public mutable fields	`S1104`	Encapsulation violation
Duplicated literals	`S1192`	Duplicate abstraction
Nested try blocks	`S1141`	Poor logic structuring
Logging using System.out	`S106`	Non-standard logging design

These code smells act as **indicators of deeper design problems**.

2.3 Subsystem-Level Code Issues

Weblog & Content Subsystem

Files with heavy code smells:

- `RomeFeedFetcher.java`
- `HTMLSanitizer.java`
- `CalendarTag.java`
- `CalendarModel.java`

Observed issues:

- High complexity
- Mixed responsibilities
- Duplicate logic

User & Role Management Subsystem

Files:

- `RollerUserDetailsService.java`
- `Manager.java`
- `Planet.java`

Issues:

- Centralized logic (hub behavior)
- Feature-heavy methods
- Generic exception handling

Search & Indexing Subsystem

Files:

- `RomeFeedFetcher.java`
- `HTMLSanitizer.java`
- `FetcherException.java`

Issues:

- Processing + validation logic mixed
- Poor modular boundaries

3. DesigniteJava Design Smell Results

Designite analysis produced **design-level architectural smells**.

Design Smell Type	Count
Unutilized Abstraction	311
Deficient Encapsulation	88
Insufficient Modularization	56
Cyclic-Dependent Modularization	49
Broken Hierarchy	35
Hub-like Modularization	3
Others	14

Total design smell instances detected: **~550+**

4. Mapping Sonar Findings to Design Smells

Sonar Evidence	Designite Smell
High complexity methods	Insufficient Modularization
Public fields	Deficient Encapsulation
Classes referenced widely	Hub-like Modularization
Mutual dependencies	Cyclic-Dependent Modularization

Duplicate literals	Broken Hierarchy / Duplicate Abstraction
Unused test structures	Unutilized Abstraction

Thus, SonarQube provides **symptoms**, while Designite confirms **architectural design flaws**.

5. Conclusion of Analysis

The system exhibits:

- **High maintainability debt**
- **Low cohesion and poor modularity**
- **Encapsulation violations**
- **Architectural coupling (cycles)**
- **Overly complex processing classes**

These findings justify refactoring aimed at:

- Improving modular structure
- Breaking cyclic dependencies
- Enhancing encapsulation
- Reducing complexity