**QA-Report**

von frag.jetzt-Backend

Name: Dang Quynh Tram, Nguyen

Matrikelnummer: 5311561

Studiengang: Bioinformatik

Das *frag.jetzt-backend*-Verzeichnis unterstürzt das Verhalten des Apps *frag.jetzt*. Seine Codes müssen nicht nur ausführbar sein, sondern auch sichergestellt werden, dass sie unbedingt keine Probleme verursachen können. Deswegen muss man die Code-Qualität analysieren. In diesem Bericht werden die Codes unter der Zulässigkeit (Reliability), der Wartbarkeit (Maintainability) und der Sicherheit (Security) bewertet. Danach werden alle Bugs fixiert und alle technischen Schulden (Kategorie: Major und Critical), die hochwahrscheinlich die Probleme auslösen, werden reduziert. Die Verbesserung der Codes nennt man Refactoring. Am Ende kommt die Sammlung der Erkenntnisse und daraus die Empfehlungen, Code-Qualität zu garantieren.

Inhaltverzeichnis

[1 Auftrag 4](#_Toc125669236)

[2 Methodik 5](#_Toc125669237)

[3 Umsetzung 6](#_Toc125669238)

[3.1 Blocker (Code Smell) 6](#_Toc125669239)

[3.2 Major 7](#_Toc125669240)

[3.3 Critical 8](#_Toc125669241)

[4 Erkenntnisse 10](#_Toc125669242)

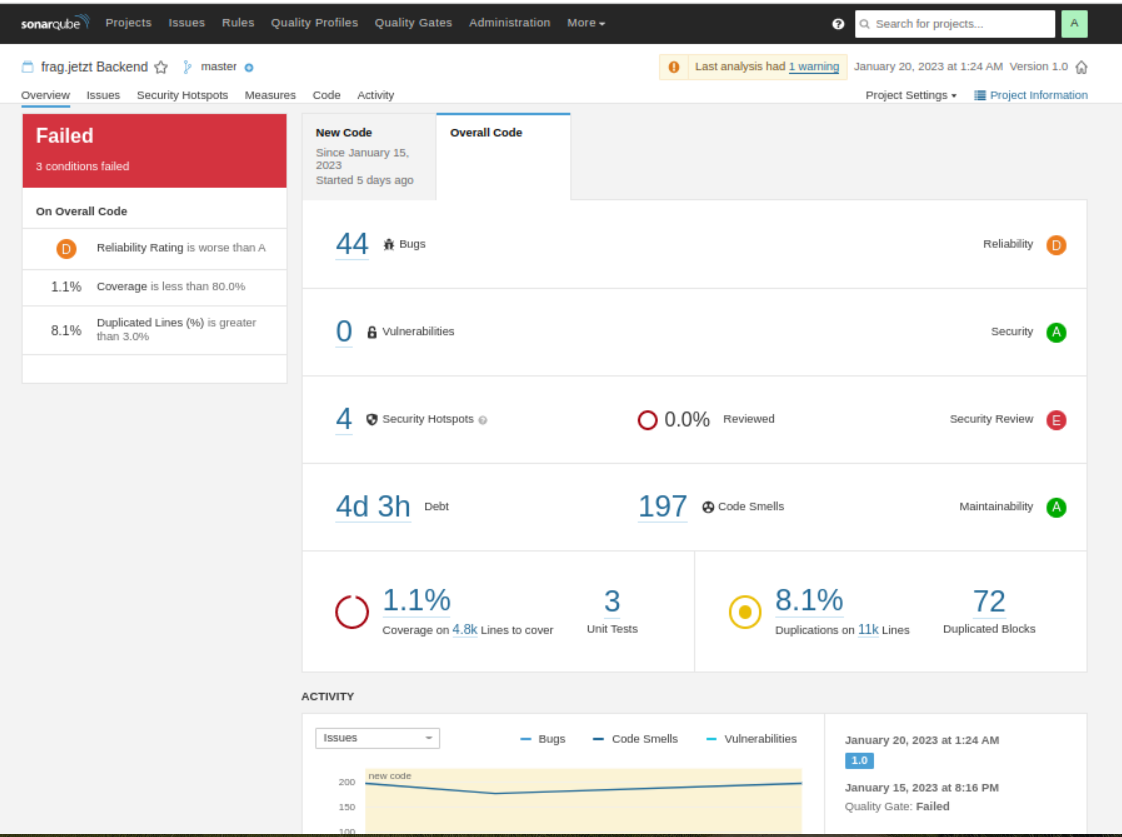
[5 Empfehlung 11](#_Toc125669243)

[6 Anhang: CNES-Reports 11](#_Toc125669244)

[7 Anhang: Glossar der technischen Begriffe 12](#_Toc125669245)

# Auftrag

Die Codes im *frag.jetzt-*backend-Verzeichnis wurden vom App Sonarqube analysiert und bewertet.



Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Durch die Analyse weißt man, dass die *frag.jetzt-backend* insgesamt 12,341 Code-Zeilen beinhaltet und sie meistens in Java geschrieben sind. Die Bewertungen der Codes sind D in der Zulässigkeit, A in der Sicherheit und der Wartbarkeit. Die Probleme sind 44 Bugs, 197 Code Smells und kein Vulnerability. Sie werden in verschiedenen Kategorien der Severity eingeordnet. Das Ziel der Refactoring ist die Verbesserung der Codes, indem die hochwahrscheinlich problematische Bugs und technische Schulden fixiert werden. Die Codes gehören zu alle Critical-Probleme, zwei Major Bugs und vier Blocker - Code Smell.

# Methodik

Durch die Code-Analysen bei Sonarqube erkennt man alle Bugs und die technischen Schulden im *frag.jetzt-backend*-Verzeichnis. Die Analysen zeigen, was die Bugs oder die Schulden sind, wo genau sie liegen, warum und wie man sie fixieren oder reduzieren sollte. Meisten Bugs und Schulden werden in Java-Dateien gefunden, deswegen kann man eine Umgebung, wo Java-Dateien lesbar ist, für Refactoring anwenden. **VS Code** wird in diesem Verfahren genutzt. Man sucht die Probleme durch die Sonarqube-Analyse in den Dateien und korrigiert die Codes.

Das Refactoring der Codes wird Schritt zu Schritt wie folgenden durchgeführt:

1. Installieren Sie das Gesamtprojekt (Frontend und Backend-Services) per Docker: [README.md · staging · ARSnova / frag.jetzt · GitLab (thm.de)](https://git.thm.de/arsnova/frag.jetzt/-/blob/staging/README.md)

2. Installieren Sie SonarQube per Docker: [analysis · staging · ARSnova / frag.jetzt Backend · GitLab (thm.de)](https://git.thm.de/arsnova/frag.jetzt-backend/-/tree/staging/analysis)

3. Aktualisieren Sie als Administrator\*in (Zugang: admin / admin) in SonarQube (Tab Marketplace) die Prüfregeln für die Sprachen Java, HTML, CSS und JS/TS.

4. Führen Sie die erste Code-Analyse durch: [Files · staging · ARSnova / frag.jetzt · GitLab (thm.de)](https://git.thm.de/arsnova/frag.jetzt/-/tree/staging#code-style-analysis)

5. Installieren Sie das CNES-Report-Plugin und erstellen Sie für die erste Code-Analyse einen CNES-Report im PDF-Format.

6. Eliminieren Sie den Critical und die zwei Major Bugs.

7. Reduzieren Sie die Technischen Schulden, indem Sie die vier Code Smells der Kategorie Blocker entfernen.

8. Testen Sie Ihre Refactorings, indem Sie die Business Use Cases aus dem Testat #2a in Ihrer frag.jetzt-Installation durchlaufen.

9. Führen Sie die zweite Code-Analyse durch: [Files · staging · ARSnova / frag.jetzt · GitLab (thm.de)](https://git.thm.de/arsnova/frag.jetzt/-/tree/staging#code-style-analysis)

10.Erstellen Sie für die zweite Code-Analyse einen CNES-Report.

# Umsetzung

Die Codes wurden auf VS Code, wo Java-Sprache unterstützt, korrigiert. Besonders speichert VS Code eine Kopie vor der Änderung, damit man 2 Versionen vergleichen kann. Die genauen Korrekturen werden in folgenden beschrieben.

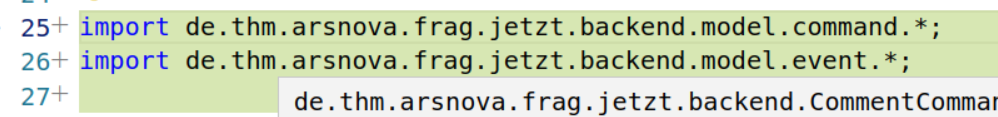
## Blocker (Code Smell)

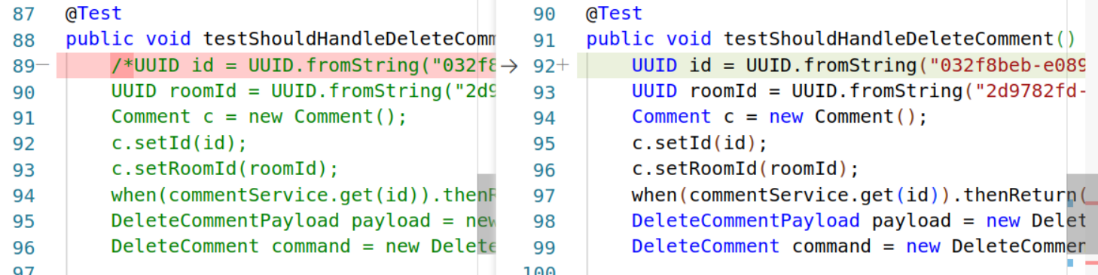
**Issue 1, 2, 3: “Add at least one assertion to this test case”**

Lösung:

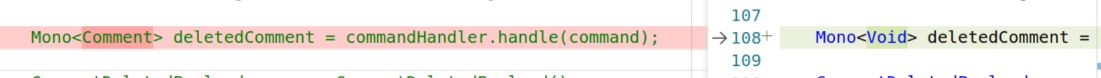
Aller Code im Test-Block wird von dem Kommentar wieder zum Code umgesetzt --> Achtung: Um den Test problemlos durchzuführen, muss man die notwendigen Pakete und Klasse zu importieren

File: src/test/java/de/thm/arsnova/frag/jetzt/backend/CommentCommandHandlerTest.java

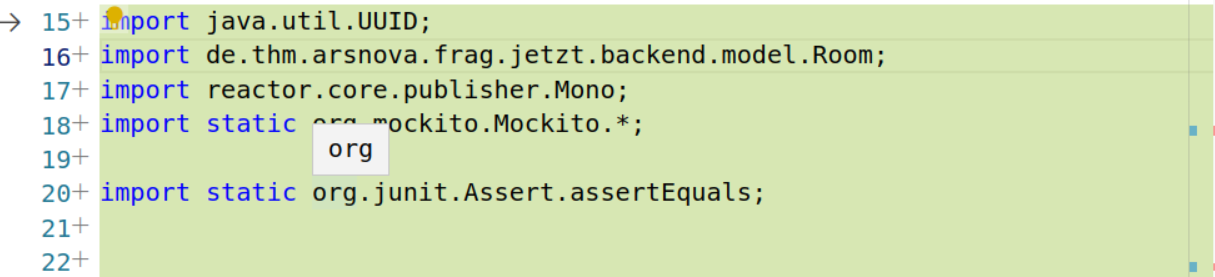


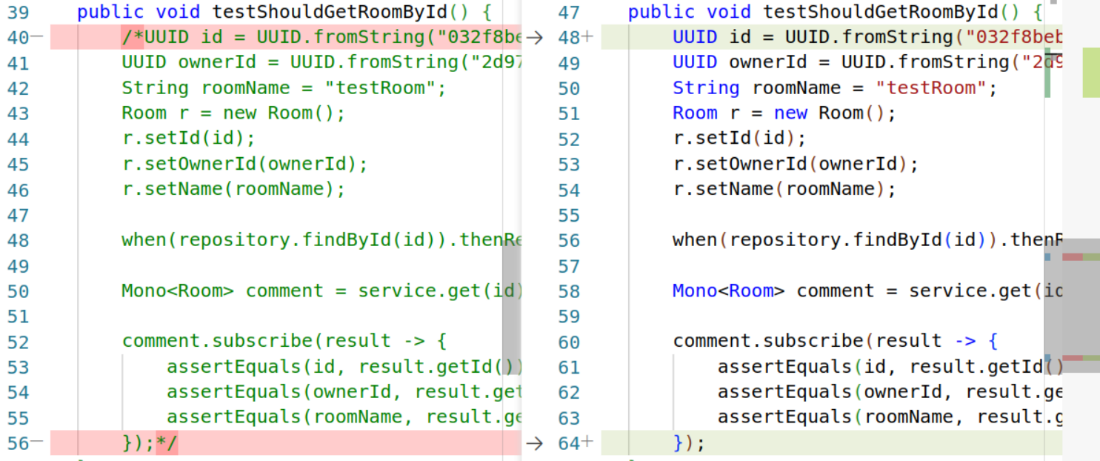


Änderung des Typs, um die Methode zu passen



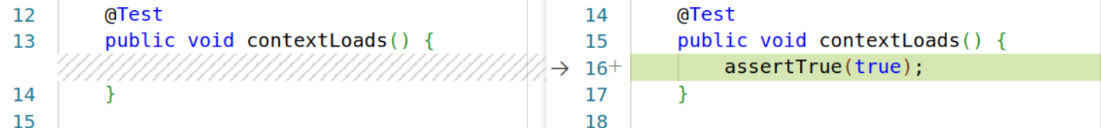
File: src/test/java/de/thm/arsnova/frag/jetzt/backend/RoomServiceTest.java





File: src/test/java/de/thm/arsnova/frag/jetzt/backend/CommentWebServiceTests.java

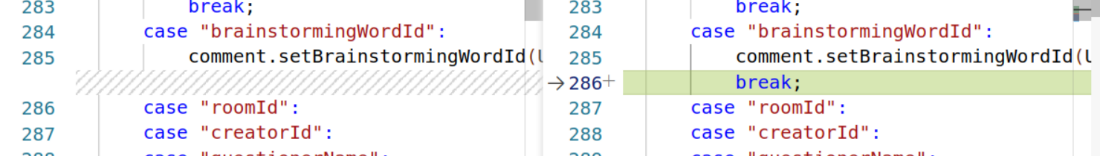
Lösung: Füge eine neutrale *assert*-Anweisung ein.



**Issue 4: “End this switch case with an unconditional break, return or throw statement”**

Lösung: *break*-Anweisung im Fall, der keinen Unterbrechung hat.

File: src/main/java/de/thm/arsnova/frag/jetzt/backend/service/CommentService.java



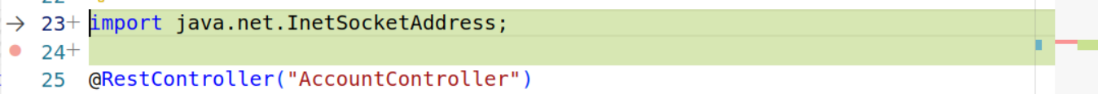
## Major

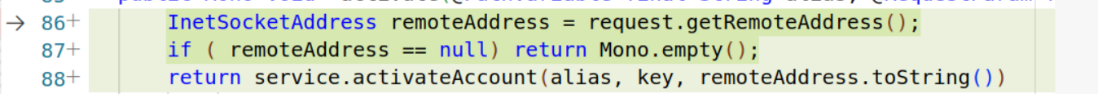
1. Code Smell

**Issue: A "NullPointerException" could be thrown; "getRemoteAddress()" can return null.**

File: src/main/java/de/thm/arsnova/frag/jetzt/backend/controller/AccountController.java

Lösung: Addieren einer Bedingung für null



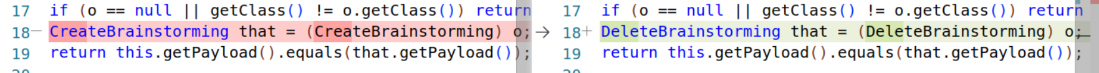


1. Bug

**Issue: Remove this call to "equals"; comparisons between unrelated types always return false.**

File: src/main/java/de/thm/arsnova/frag/jetzt/backend/model/command/DeleteBrainstorming.java

Lösung: Änderung des Objekt-Typen



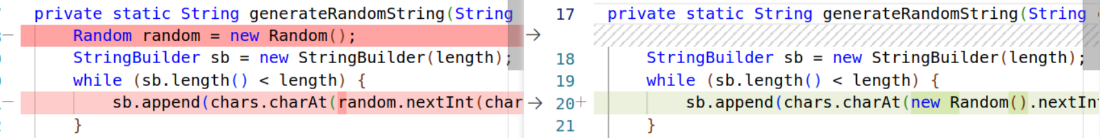
## Critical

1. Bug

**Issue: Save and re-use this "Random".**

File: src/main/java/de/thm/arsnova/frag/jetzt/backend/util/RandomUtils.java

Lösung: Nutzen *new* *Random()*, statt eine Variable Typ *Random*



1. Code Smell

**Issue 1: Use already-defined constant 'ROOM\_SUBSCRIPTION\_MAPPING' instead of duplicating its value here.**

File: src/main/java/de/thm/arsnova/frag/jetzt/backend/controller/CommentChangeController.java

Lösung: Ersetzen den Wert durch die Variable, der denselben Wert definiert



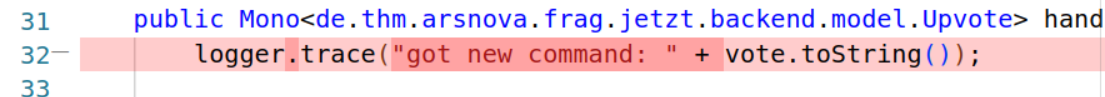
Erkenntnis: Nutzen die Variable, die den Wert definiert, statt denselben Wert

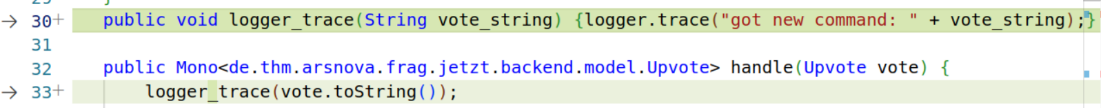
**Issue 2, 3, 4, 5, 14: Define a constant instead of duplicating this literal "got new command: " many times.**

File: alle Files im Paket src/main/java/de/thm/arsnova/frag/jetzt/backend/handler/ und src/main/java/de/thm/arsnova/frag/jetzt/backend/service/CommentService.java

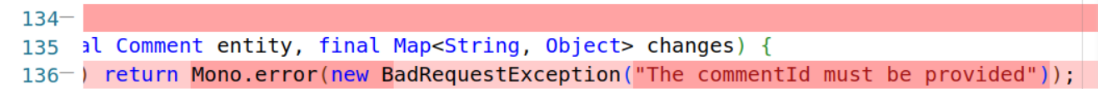
Lösung: Definieren eine Methode, die die wiederholende Anweisung ausführt

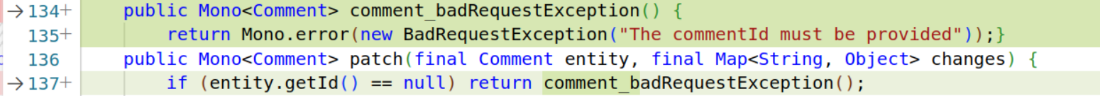
z. B. im File: VoteCommandHandler.java





File: CommentService.java



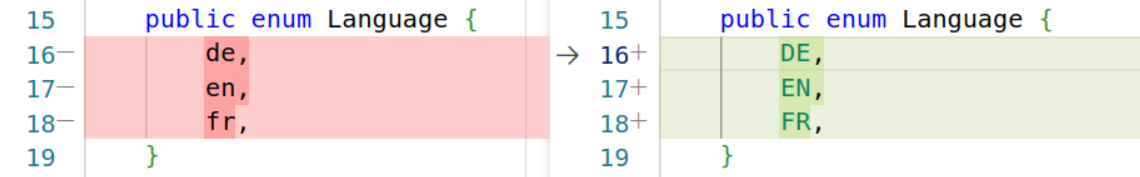


(gleiche Lösung in anderen Files)

**Issue 6, 7, 8: Rename this constant name to match the regular expression '^[A-****Z][A-Z0-****9]\*(\_[A-Z0-****9]+)\*$'.**

File: src/main/java/de/thm/arsnova/frag/jetzt/backend/controller/CommentChangeController.java

Lösung: Großschrift für konstante Namen

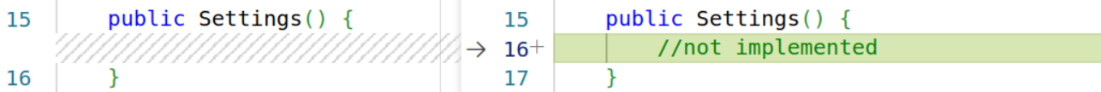


**Issue 9, 11, 12: Add a nested comment explaining why this method is empty, throw an UnsupportedOperationException or complete the implementation.**

File: src/main/java/de/thm/arsnova/frag/jetzt/backend/model/Settings.java, src/main/java/de/thm/arsnova/frag/jetzt/backend/model/event/RoomCreatedPayload.java und src/main/java/de/thm/arsnova/frag/jetzt/backend/model/event/VoteCreatedPayload.java

Lösung: Schreiben ein Kommentar

File: Settings.java

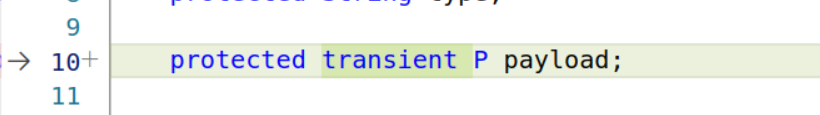


(ähnliche Lösung für andere Files)

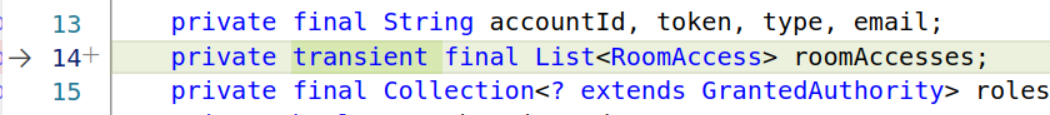
**Issue 10, 13: Make "payload"/"roomAccesses" transient or serializable.**

Lösung: Setzen die Variable *transient*

File: src/main/java/de/thm/arsnova/frag/jetzt/backend/model/WebSocketMessage.java



File: src/main/java/de/thm/arsnova/frag/jetzt/backend/security/AuthenticateUser.java



**Issue 15, 16: Refactor this method to reduce its Cognitive Complexity from 20 to the 15 allowed.**

File: src/main/java/de/thm/arsnova/frag/jetzt/backend/service/RoomService.java und src/main/java/de/thm/arsnova/frag/jetzt/backend/service/CommentService.java

Lösung: Setzten die Status ‘Resolved as Won’t Fix’

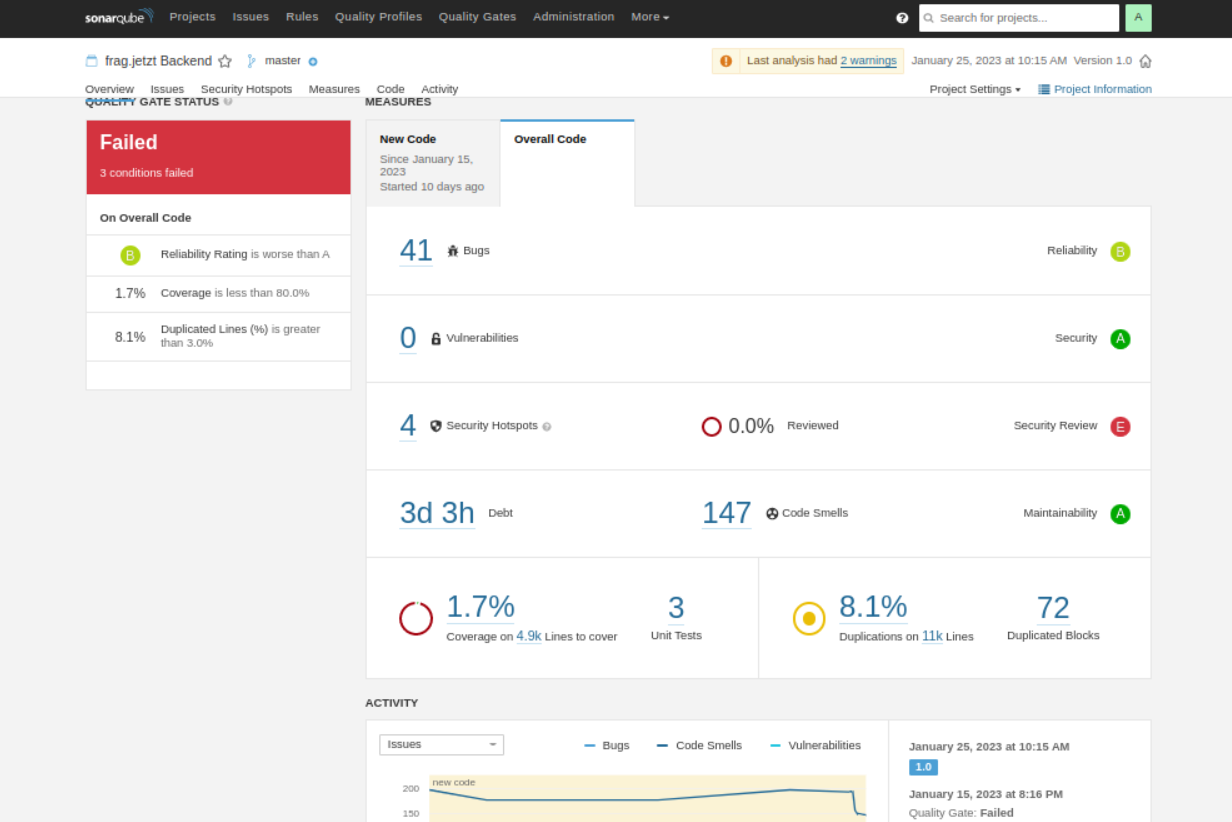
Erkenntnis: Die kognitive Komplexität tritt auf aber kann nicht verbessert werden

# Erkenntnisse

Die genauen Erkenntnisse bei dem Refactoring der Codefragmente:

|  |  |  |  |
| --- | --- | --- | --- |
| **Severity** | **Typ** | **Issue** | **Erkenntnis** |
| Blocker | Code Smell | 1, 2, 3 | *Test*-Block sollte mindestens eine *assert*-Anweisung enthalten, damit der Test sinnvoll durchgeführt wird. |
| 4 | Wenn der Fall von *switch-case* nicht endet, können die Anweisungen darin unendlich ausgeführt werden. |
| Major | Code Smell |  | Der Parameter einer Methode sollte nicht Null sein. |
| Bug |  | Sinnlose lange Anweisung sollte vermeidet werden.  Beim Vergleich der Variablen sollten ihre Typen identisch sein. |
| Critical | Bug |  | Eine Variable, die *new Random()* definiert wird, sichert nicht ab, dass sein Wert in jedem Aufruf zufällig ist. |
| Code Smell | 1 | Wenn ein Wert als eine Variable definiert, sollte man diese Variable aufrufen, statt denselben Werts. |
| 2, 3, 4, 5, 14 | Wenn eine Anweisung vielmals angewendet wird, kann man sie als eine *void*-Methode definieren und bei der Verwendung die Methode aufrufen. |
| 6, 7, 8 | Die Namen der Konstanten müssen die Regel des regulären Ausdruckes erfolgen: in Großbuchstaben geschrieben. |
| 9, 11, 12 | Eine Methode-Block sollte nicht leer sein. |
| 10, 13 | Felder in einer „Serializable“-Klasse sollten entweder *transient* oder serialisierbar sein. |
| 15, 16 | Fortlaufende Nutzung der *if-else*-Bedingung löst manchmal die kognitive Komplexität aus, aber man kann nicht anders machen. |

Nach dem Bugfixieren und der Reduzierung der technischen Schulden zeigt die Sonarqube-Analyse so an:



Man kann sehen, dass die Zuverlässigkeit (Reliability) von D auf B verbessert wird, dabei 3 Bugs fixiert werden. Die Anzahl von Code Smell sinkt ab. Da folgt, dass die Zeit für Behebung von Code Smell kürzer bewertet wird.

# Empfehlung

Um die Qualität der Codes sicherzustellen, sollte man beim Codeschreiben die Syntax und die Regel der Programmiersprache richtig achten. Die Duplikation, unlogische Anweisungen oder unnötige Variablen sollte man auch vermeiden. In diesem Fall je kurzer man schreibt, desto besser ist die Codes.

Viele Programmiersprache-Umgebungen unterstürzen, Bugs zu detektieren und zu erheben. Empfehlung für die Java-Umgebung: Intellji IDEA, Eclipse. Wenn die Umgebung hilflos bei der Sicherung der Code-Qualität ist, kann man gleichzeitig beim Schreiben Code analysieren lassen. Sonarqube ist dafür empfohlen.

# Anhang: CNES-Reports

CNES-Report vor dem Refactoring: <https://drive.google.com/file/d/1ddnc7URK5uJIhT_sYSLMAjQfg_UXO7am/view?usp=share_link>

CNES-Report nach dem Refactoring: <https://drive.google.com/file/d/18gqWWa09Zw0bwf_NVKst-98FEN8R4vyK/view?usp=sharing>

# Anhang: Glossar der technischen Begriffe

<https://drive.google.com/file/d/1MsahNQw4vwlAxjNluCsx0br9pLSsIpur/view?usp=sharing>