

SQE3-18ILV

SQElevator Project

Team D

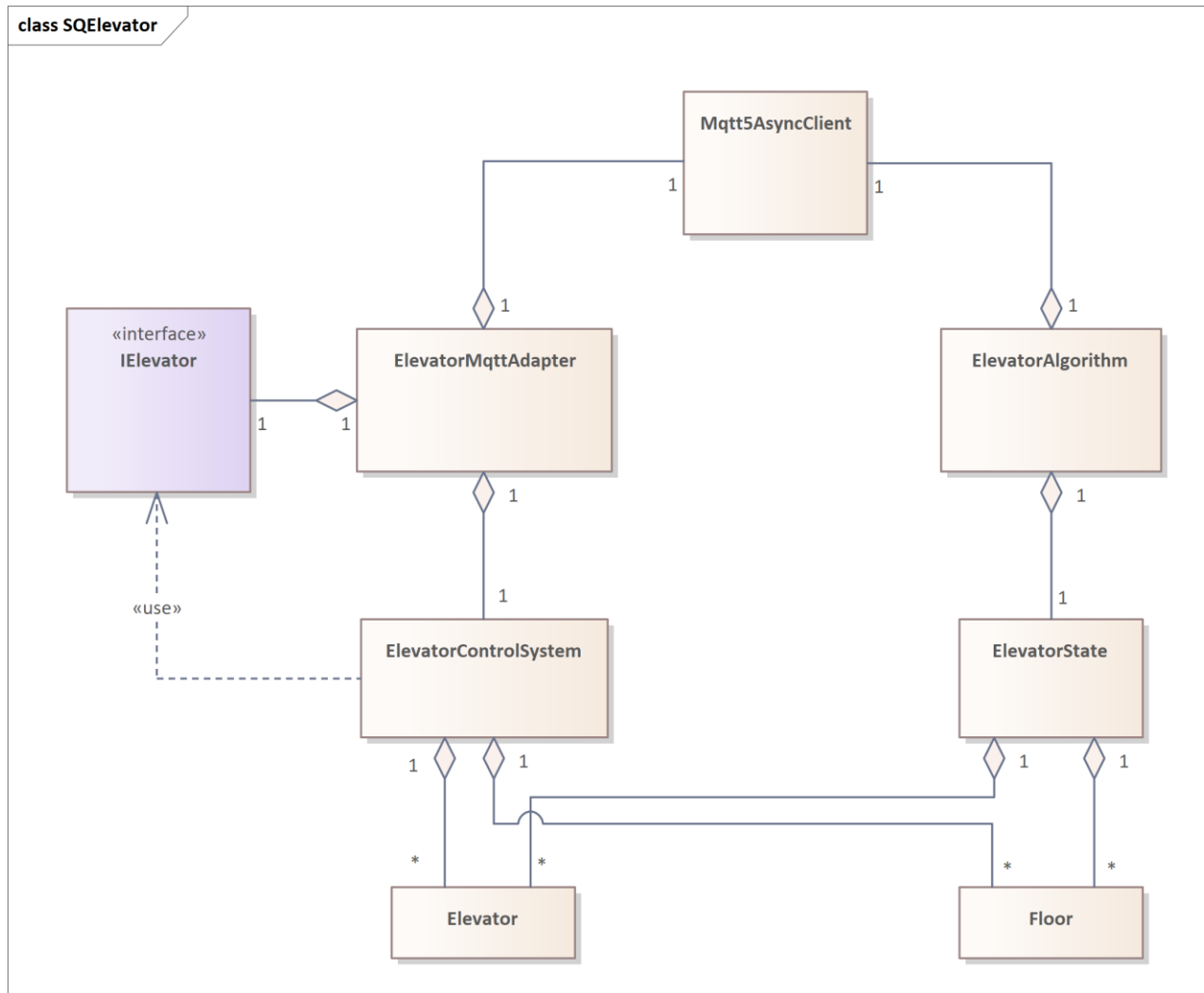
© Damianschitz, Oberndorfer, Reinberger



Struktur

- Software-Architektur
- Live-Demonstration
- Projekt- und Produktqualität
- Tieferer Einblick

Software-Architektur



Live-Demonstration



Projekt- und Produktqualität – 1

- GitHub Actions
 - Automatisiertes Testen mittels **Maven**™ bei jedem
 - Pull Request
 - Push auf master



-  für Code Coverage
 - Build-Goal während `mvn test`
- Javadoc
 - Build-Goal `mvn javadoc:javadoc`


```
build summary

JaCoCo Test Coverage Summary


• Coverage: 87.151%
• Branches: 86.071%
• Generated by: jacoco-badge-generator


Job summary generated at run-time
```

Projekt- und Produktqualität – 2


- Statische Analyse mittels 
 - Ebenfalls Teil der GitHub-CI-Pipeline bei jedem
 - Pull Request
 - Push auf master


No new bugs are introduced	Reliability rating is A
No new vulnerabilities are introduced	Security rating is A
New code has limited technical debt	Maintainability rating is A
All new security hotspots are reviewed	
New code has sufficient test coverage	Coverage is greater than or equal to 80.0% ?
New code has limited duplications	Duplicated Lines (%) is less than or equal to 3.0% ?


 **elevator-mqtt-team-d** PUBLIC


 **Passed**


Last analysis: 07/01/2025, 23:14 • 1.2k Lines of Code • Java, XML

 **0**
Security

 **0**
Reliability

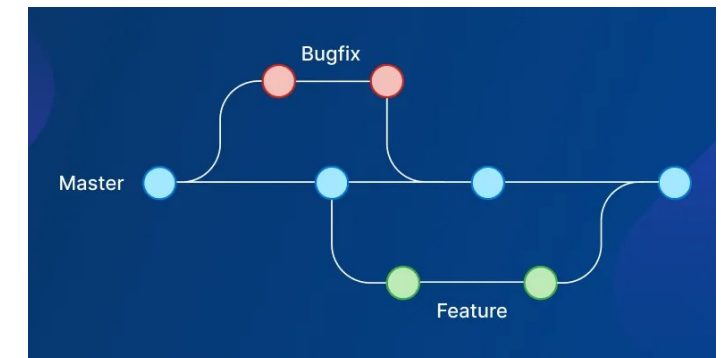
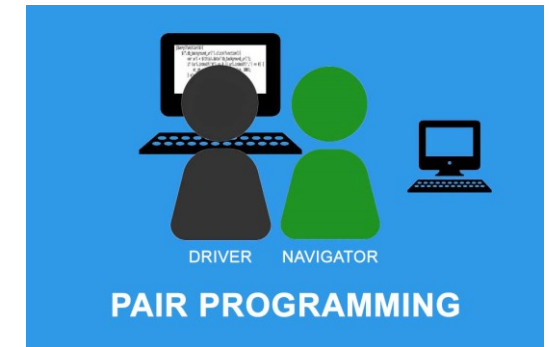
 **0**
Maintainability

 **100%**
Hotspots Reviewed

 **1.5%**
Duplications

Projekt- und Produktqualität – 3

- Code Reviews
 - Integrierter Prozess bei jedem GitHub Pull Request
 - Zusätzlich wurde 90% der Zeit im Pair-Programming entwickelt
- Feature-Based-Workflow
 - Entwickeln auf feature Branches
 - Tests werden direkt mitimplementiert
- Issue-Management
 - Aufgrund Pair-Programming-Ansatz nicht notwendig




Projekt- und Produktqualität – 4

- Testpraktiken

- White-Box-Tests
reine Datenklassen
100% Coverage möglich und sinnvoll
- Black-Box-Tests
Klassen mit Testcontainer und/oder Mocks
100% Coverage unmöglich erreichbar
(z.B. „unerreichbares“ default in switch)
- Manueller Gesamttest
- Einsatz einer Test Suite

```
package at.fhhagenberg.sqelevator;  
  
import at.fhhagenberg.sqelevator.adapter.*;  
import at.fhhagenberg.sqelevator.algorithm.*;  
  
import org.junit.platform.suite.api.SelectClasses;  
import org.junit.platform.suite.api.Suite;  
  
/**  
 * Test suite for the elevator system  
 */  
@Suite  
@SelectClasses({ ... })  
public class ElevatorTestSuite {}
```


„Lessons Learned“

- Testen ist meist zeitintensiver als die Implementierung selbst
- Dependency-Versionierungs-Mismatch bei Hive MQ Testcontainer
-  hat seine Eigenheiten, vor allem wenn man C++ gewohnt ist
- Multithreading erfordert zusätzliche Vorsicht beim Testprozess

Weird Problem ☹️

- Test von RMI Exception führt zu eigenartigem Verhalten
 - Test läuft durch, wenn man ihn in IntelliJ separat ausführt
 - Test läuft durch, wenn man gesamte Testklasse in IntelliJ separat ausführt
 - Test läuft durch, wenn man gesamte Test Suite in IntelliJ ausführt
 - Test läuft (gelegentlich) durch, wenn man Maven-Test in IntelliJ ausführt
 - Test läuft (noch seltener) durch, wenn man Maven-Test in der Pipeline ausführt
- Fehler:
 - MQTT Connect schlägt fehl
 - Exception wird nicht in vorgegebener Zeit behandelt
 - Black Magic?

