

# **ANALYSE & EVALUIERUNG DER SOFTWAREQUALITÄT DES PROJEKTS UIS**

## **GROUP 11**

**Cordula Eggerth (00750081)**

**Armin Hasler (00725841)**

**Franz Kienegger (01426716)**

VU Software Quality Management  
Sommersemester 2019

# Agenda

1. Vorgeschlagene Qualitätsmanagement-Pipeline
2. Status Quo des Projekts bzgl. Qualität und Verbesserungsvorschläge
3. Conclusio

# Agenda

1. **Vorgeschlagene Qualitätsmanagement-Pipeline**
2. Status Quo des Projekts bzgl. Qualität und Verbesserungsvorschläge
3. Conclusio

## Qualitätsmanagement-Pipeline und -Tools

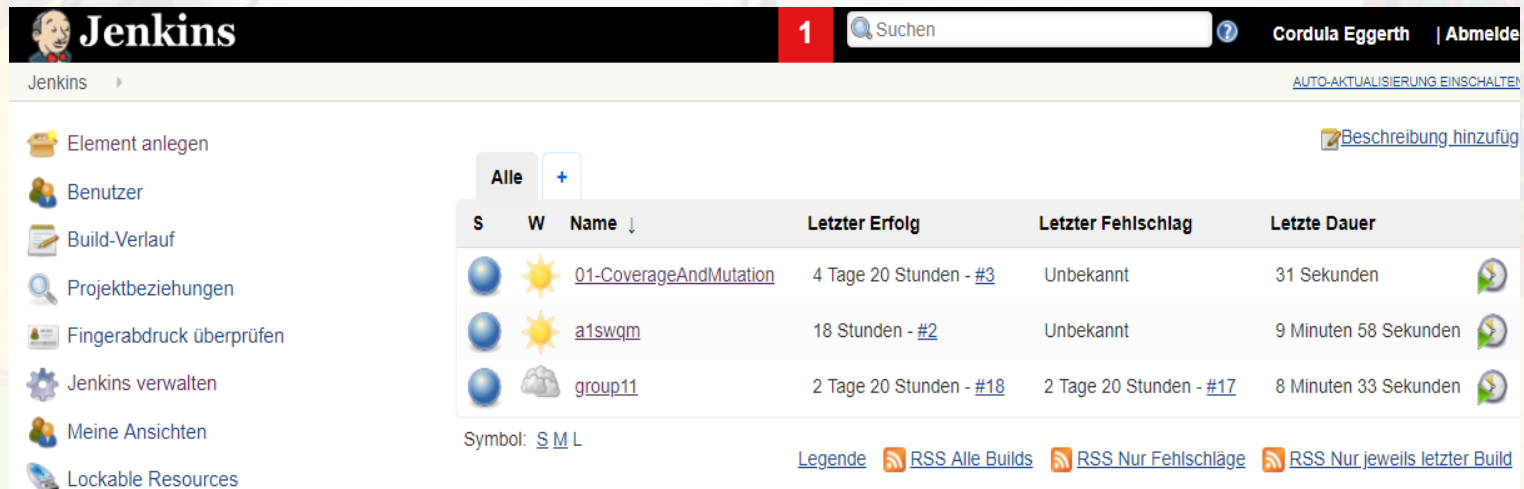
- **Jenkins:** Continuous Integration
- **SonarQube:** Code- und Qualitätsanalyse
- **JaCoCo:** Coverage-Report
- **CheckStyle:** Statische Code-Analyse und Coding Styles
- **SonarLint:** Überprüfung von Coding Styles / Standards

### Zur Konfiguration:

- **JaCoCo, CheckStyle:** jeweils als *dependency* im pom.xml hinzugefügt
- **Jenkins, SonarQube:** Einbindung mittels Docker (*services* in docker-compose.yml File)

## Gründe für die Verwendung von Jenkins

- Bietet zahlreiche hilfreiche Plugins (u.a. SonarQube Scanner, JaCoCo-Integration, SonarLint-Verbindung)
- Kostenloses CI-Tool
- Direkte Verbindung mit dem zugehörigen Git Repository möglich (aber auch Verwendung von File-System als Basis)
- Flexibel gestaltbar je nach den Bedürfnissen des Projekts



The screenshot shows the Jenkins web interface. The top navigation bar includes the Jenkins logo, a search bar, and the user name 'Cordula Eggerth' with a 'Abmelden' (Logout) link. The left sidebar contains a menu with links to 'Element anlegen', 'Benutzer', 'Build-Verlauf', 'Projektbeziehungen', 'Fingerabdruck überprüfen', 'Jenkins verwalten', 'Meine Ansichten', and 'Lockable Resources'. The main content area displays a table of builds. The table has columns for 'S' (Status), 'W' (Workspace), 'Name', 'Letzter Erfolg' (Last Success), 'Letzter Fehlschlag' (Last Failure), and 'Letzte Dauer' (Last Duration). The table lists three builds: '01-CoverageAndMutation', 'a1swqm', and 'group11'. Each build row includes a status icon (blue circle with a white dot), a workspace icon (yellow sun), and a duration icon (green circle with a white dot). Below the table, there is a 'Symbol: S M L' section and a 'Legende' (Legend) section with links for 'RSS Alle Builds', 'RSS Nur Fehlschläge', and 'RSS Nur jeweils letzter Build'.

S	W	Name ↓	Letzter Erfolg	Letzter Fehlschlag	Letzte Dauer
		<a href="#">01-CoverageAndMutation</a>	4 Tage 20 Stunden - <a href="#">#3</a>	Unbekannt	31 Sekunden
		<a href="#">a1swqm</a>	18 Stunden - <a href="#">#2</a>	Unbekannt	9 Minuten 58 Sekunden
		<a href="#">group11</a>	2 Tage 20 Stunden - <a href="#">#18</a>	2 Tage 20 Stunden - <a href="#">#17</a>	8 Minuten 33 Sekunden

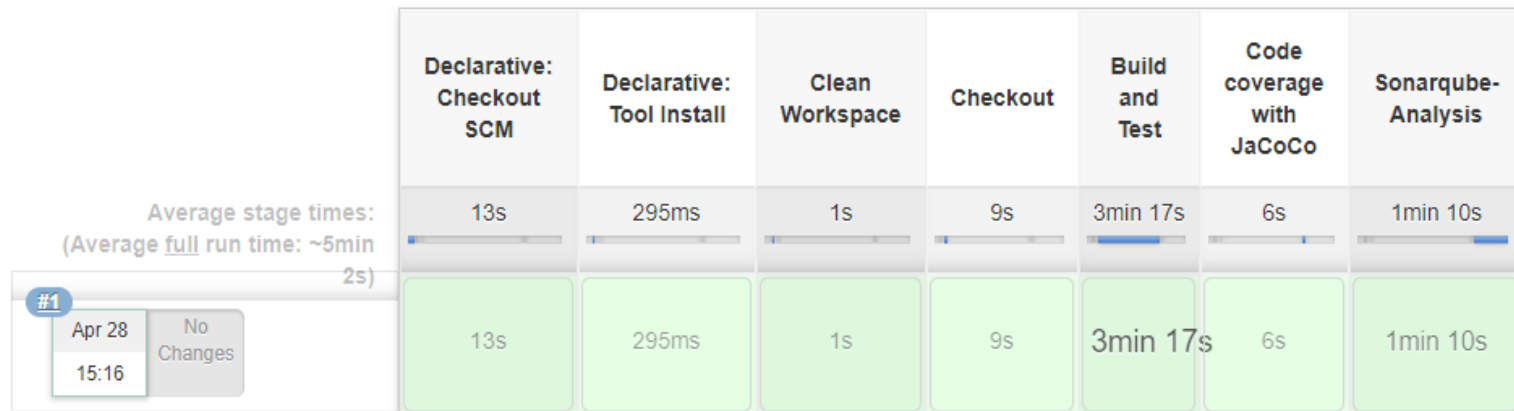
Symbol: S M L

Legende [RSS Alle Builds](#) [RSS Nur Fehlschläge](#) [RSS Nur jeweils letzter Build](#)

## Jenkins Build Pipeline

- Build-Prozess in mehrere Schritte (*stages*) geteilt
- Groovy Script, um die Pipeline anzulegen und zu steuern

### Stage View



### SonarQube Quality Gate

UIS **OK**

server-side processing: **Success**

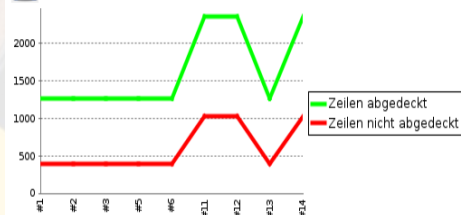


## Gründe für die Verwendung von JaCoCo

- Häufig verwendetes Open-Source-Tool mit regelmäßigen Updates
- Umfassende Test-Coverage-Analyse auf verschiedenen Granularitätsstufen
  - ✓ Kommunikation der Ergebnisse an unterschiedliche Stakeholder möglich
  - ✓ Stellen mit Bedarf für weitere Tests leicht erkennbar

### JaCoCo Coverage Report

[Download jacoco.exec binary coverage file](#)



### Overall Coverage Summary

Name	instruction	branch	complexity	Zeilen	Methoden	Klassen
all classes	57% <div><div></div></div> M: 10204 C: 13263	41% <div><div></div></div> M: 646 C: 454	58% <div><div></div></div> M: 593 C: 816	69% <div><div></div></div> M: 1035 C: 2352	81% <div><div></div></div> M: 164 C: 695	95% <div><div></div></div> M: 6 C: 121

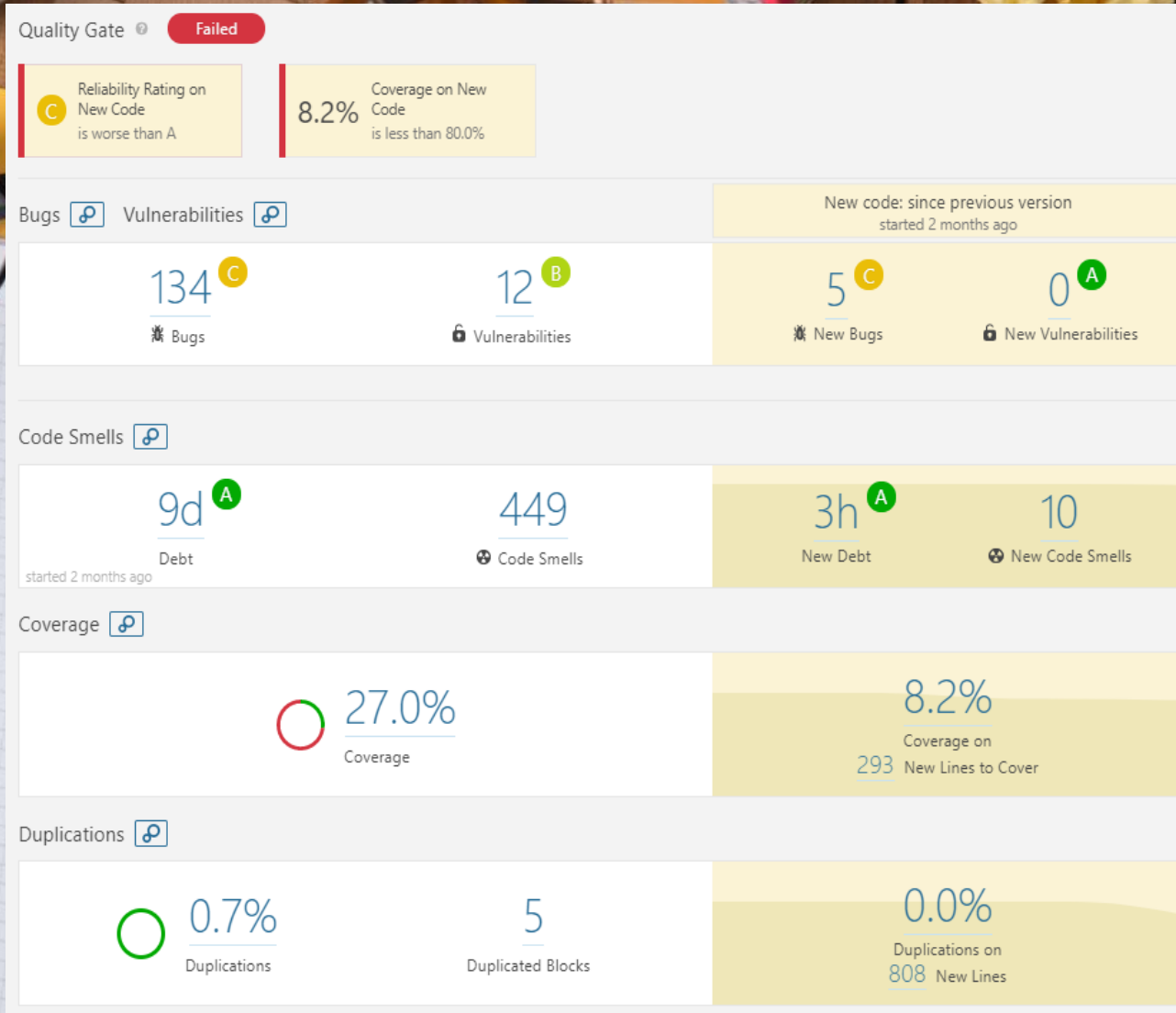
### Coverage Breakdown by Package

Name	instruction	branch	complexity	Zeilen	Methoden	Klassen
<a href="#">at.ac.tuwien.inso.sgm</a>	M: 13 C: 7 35% <div><div></div></div>	M: 0 C: 0 100% <div><div></div></div>	M: 1 C: 2 67% <div><div></div></div>	M: 6 C: 2 25% <div><div></div></div>	M: 1 C: 2 67% <div><div></div></div>	M: 0 C: 1 100% <div><div></div></div>
<a href="#">at.ac.tuwien.inso.sgm.config</a>	M: 0 C: 199 100% <div><div></div></div>	M: 0 C: 0 100% <div><div></div></div>	M: 0 C: 9 100% <div><div></div></div>	M: 0 C: 46 100% <div><div></div></div>	M: 0 C: 9 100% <div><div></div></div>	M: 0 C: 3 100% <div><div></div></div>
<a href="#">at.ac.tuwien.inso.sgm.controller</a>	M: 219 C: 193 47% <div><div></div></div>	M: 26 C: 2 7% <div><div></div></div>	M: 19 C: 15 44% <div><div></div></div>	M: 56 C: 47 46% <div><div></div></div>	M: 6 C: 14 70% <div><div></div></div>	M: 1 C: 4 80% <div><div></div></div>

source: [https://www.123rf.com/photo\\_99857951\\_top-view-of-office-desk-work-with-computer-supplies-keyboard-graphic-tablet-on-wooden-table-creative.html](https://www.123rf.com/photo_99857951_top-view-of-office-desk-work-with-computer-supplies-keyboard-graphic-tablet-on-wooden-table-creative.html)

## SonarQube

- Reliability Rating für bestehenden/ neuen Code
- Information über Technical Debt (in Zeiteinheiten zur Behebung)
- Übersicht über Bugs, Code Smells, Coverage und Duplications





## SonarLint

- **Einbindung von SonarLint in IntelliJ und in den SonarQube-Server**
  - ✓ Qualitätsregeln können direkt aus SonarQube-Server gelesen werden
  - ✓ Entwickler können direkt beim Schreiben des Codes auf mögliche Fehler/Verletzungen der Qualitätsregeln hingewiesen werden (ohne auf den Abschluss des Builds am SonarQube-Server zu warten)
- **Statische Code-Analyse und Förderung eines einheitlichen Code-Stils**

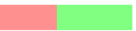
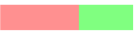
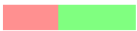
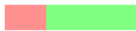
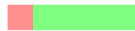
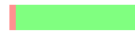
# Agenda

1. Vorgeschlagene Qualitätsmanagement-Pipeline
- 2. Status Quo des Projekts bzgl. Qualität und Verbesserungsvorschläge**
3. Conclusio

## Coverage (gemäß JaCoCo-Report)

- Testabdeckung erscheint auf den ersten Blick angemessen, könnte aber besser sein (→ Ziel von 80%; z.B. Build Failure in Jenkins als Maßnahme, falls nicht erfüllt):


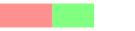




Overall Coverage Summary

Name	instruction	branch	complexity	Zeilen	Methoden	Klassen
all classes	57%  M: 10204 C: 13263	41%  M: 646 C: 454	58%  M: 593 C: 816	69%  M: 1035 C: 2352	81%  M: 164 C: 695	95%  M: 6 C: 121

- Klassen, die eigentlich nicht getestet werden müssen, wurden zunächst berücksichtigt:
  - z.B. DataInitializer, DemoDataInitConfig, package initializer

- Positiveres Bild nach Ausschluss der oben genannten Klassen:

Overall Coverage Summary

Name	instruction	branch	complexity	Zeilen	Methoden	Klassen
all classes	80%  M: 3200 C: 13194	42%  M: 628 C: 454	60%  M: 547 C: 815	81%  M: 543 C: 2351	85%  M: 127 C: 694	96%  M: 5 C: 120

- **Anmerkung:** Ob die Coverage insgesamt “ausreichend” ist, ist davon abhängig, welches der obigen sechs Kriterien der Testabdeckung betrachtet wird



# Empfehlungen zur Verbesserung der Coverage

- Verbesserung der Coverage bzgl. *branch* (42%) und *complexity* (60%)
- Erhöhung der Testabdeckung im Modul *domain* bzw. eventuell Ausschluss aus JaCoCo (da v.a. Getter & Setter der DTOs und Entities betroffen)
- Erhöhung der Testabdeckung der *services* und *Controller*, die zentrale Komponenten der Anwendung sind
- Vollständige Abdeckung der `equals ()`-Methoden

# Statische Codeanalyse (mittels SonarQube)

### ■ Ergebnisse des ersten Durchlauf mit SonarQube:



- Negativ: Hinweis auf wenig sorgfältige Programmierung
- Positiv: Sehr geringe Codeduplikation (Ziel: 0%)

### ■ Technical Debt:

- **Vulnerabilities**: 3h 10min → hohe Priorität, können zu Anomalien führen
- **Bugs**: 1d 3h → einige “False Positives” (z.B. in DataInitializers)
- **Code Smells**: 9d → sollten behoben werden, da Lesbarkeit und Refactoring-Möglichkeiten sonst leiden könnten



# Kennzahlen gemäß Analyse mit SonarQube

## Size: (Umfang des Projekts)

Lines of Code	8,089
Lines	11,207
Statements	2,703
Functions	801
Classes	151
Files	150
Comment Lines	494

## Complexity: \*

Cyclomatic Complexity	1,207
Cognitive Complexity	535

## Duplications: (Codeverdoppelung)

### Overall

Density	0.8%
Duplicated Lines	85
Duplicated Blocks	5
Duplicated Files	3

### \* Anmerkung:

- Cyclomatic Complexity: wieviele verschiedene Pfade es durch ein Projekt gibt in der betrachteten Granularitätsstufe
- Cognitive Complexity: wie verständlich der Code für Menschen ist bzw. wie er von Menschen geistig strukturiert werden würde

## Bad Quality Hotspots (basierend auf SonarQube-Analyse)

- Im Module `backend` befinden sich die meisten Bugs und Code Smells
  - sollten umgehen bearbeitet werden
  - `initializer` und `service` fallen als Hotspots schlechter Code-Qualität auf

	Lines of Code	Bugs	Vulnerabilities	Code Smells	Coverage
backend/src/main/java/at/ac/tuwien/inso/sqm	3,818	130	6	331	20.1%
initializer	1,564	126	0	267	0.0%
repository	258	0	0	3	0.0%
service	1,795	4	6	58	35.9%
utils	9	0	0	2	0.0%
validator	192	0	0	1	33.2%

# CheckStyle

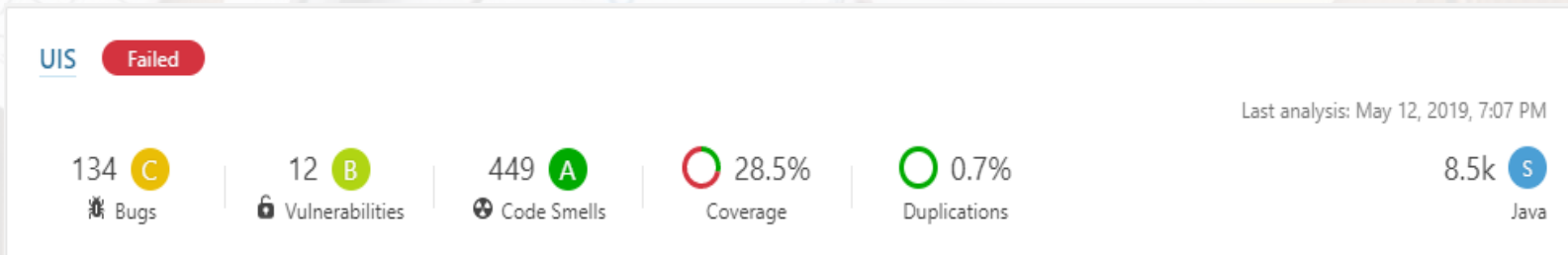
- **Basis:** Google-Checks (weil kontinuierlich angepasst) – jedoch mit Anpassungen
- **Strategie:** Build Failure, falls CheckStyle-Regeln nicht erfüllt (*severity: error*)
  - ✓ Code-Qualität nachhaltig steigern und Code-Style vereinheitlichen
- **Verbleibende Verstöße nach Anpassung der CheckStyle-Regeln:**

Checkstyle-Regel	Anzahl von Verstößen
AnnotationLocation	1
LeftCurly	2
NeedBraces	106
RightCurly	35
OneStatementPerLine	1
AvoidStarImport	83
Indentation	506
LambdaParameterName	1
MemberName	7
LineLength	131
EmptyLineSeparator	3
MethodParamPad	2
OperatorWrap	3
ParenPad	1
WhitespaceAround	325



# Qualitätsverbesserungen durch CheckStyle

- Schrittweise Einführung von CheckStyle-Regeln → geordnete Behebung
- (Kleine) Verringerung der Code Smells (von 463 auf 449) durch Einsatz von CheckStyle



# Zusammenfassung der durch die Qualitätspipeline identifizierten Probleme und darauf basierende Ratschläge

### Bisherige Probleme im Projekt UIS:

- Hohe Anzahl an Code Smells, die die Lesbarkeit stark beeinträchtigen
- Viele Todo-Markierungen
- Wenig sorgfältige Programmierung
- Viele Rechtschreibfehler und Uneinheitlichkeit bezüglich Benennung
- Unzureichende/fehlerhafte Dokumentation
- Bisher kein Einsatz von Qualitätsanalyse-Tools

### Ratschläge:

- ✓ Einheitlicher Ansatz zur Verstärkung der Aufmerksamkeit auf Qualität
- ✓ Development Guidelines, strukturierte Code Reviews
- ✓ Knowledge Management hinsichtlich Qualitätsbelangen



# Development Guidelines für das Projektteam

- **Englisch als Projektsprache**
- **Javadoc für die öffentlichen Schnittstellen/API**
- **Kein auskommentierter Code**
  - Löschen oder tatsächlich als Code verwenden
- **Rechtschreibung überprüfen**
- **Namensgebung verbessern**
  - Einheitliche Richtlinien für Variablen, Interfaces, Klassen, etc.
  - Sinngemäße, selbsterklärende Namen
- **Log-Levels (DEBUG, INFO, WARN, ERROR) passend einsetzen**

# Branching-Strategie

- **Basis:** GitLab Flow
- Feature Branches direkt vom Master erstellt und dahin gemerged
- Feature Branches benötigen Merge Request und Review (durch anderen Developer) für Merge in Master
- **Vorteile:**
  - ✓ Modularisierung
  - ✓ Einbringen von Feedback in den Development-Prozess
  - ✓ Lerneffekt für neue Projektmitglieder



# Ausschnitt aus der strukturierten Checkliste für Code-Reviews

Zu überprüfender Punkt	Erfüllt	Mangel (Beschreibung)
Ist der Code gut strukturiert?		
Ist der Code leicht verständlich?		
Ist der Code modular aufgebaut?		
Ist der Code frei von Duplikationen und Redundanzen?		
Ist der Code frei von Sicherheitsmängeln?		
Ist die Testabdeckung ausreichend?		
Wurden Unit- und Integration-Tests implementiert?		
Werden die definierten Namenskonventionen eingehalten?		
Wurde ein etwaiges Change Log angepasst?		

# Gründe für Knowledge Management mittels GitLabWiki

- **Interaktive Weiterentwicklung des GitLabWiki während dem laufenden Softwareprojekt**
- **Direkte Verbindung des Wiki mit dem GitLab-Projekt**
- **Schneller Zugriff auf das Wiki vom GitLab-Projekt aus**
- **Kostenlose Nutzung**
- **Einbinden von Code in verschiedenen Programmiersprachen gut möglich**

source: [https://www.123rf.com/photo\\_99857951\\_top-view-of-office-desk-work-with-computer-supplies-keyboard-graphic-tablet-on-wooden-table\\_creative.html](https://www.123rf.com/photo_99857951_top-view-of-office-desk-work-with-computer-supplies-keyboard-graphic-tablet-on-wooden-table_creative.html)

# Ausschnitt aus dem vorgeschlagenen GitLabWiki (“Home Page“)

The screenshot shows the GitLab Wiki interface. The top navigation bar includes the GitLab logo, 'Projects', 'Groups', 'More', and a search bar. The main content area is titled 'Knowledge Management for Project UIS (group 11)'. On the left, a sidebar shows a tree view with icons for home, documents, and users. The main content is divided into sections: 'Contents:', '1. Development Guidelines', '2. Handling of Merge Requests and Protected Branches', '3. Quality Improvement', and '4. Quality Management and Analysis Tool Set-Up in the Project'. Each section contains a list of links to specific documents. On the right, a 'Clone repository' section lists various tools and strategies like 'Acceptance of merge requests', 'Branching strategy', 'CheckStyle', 'Code review checklist', 'Common naming patterns for groups of artifacts', 'Creation of issues', 'Dealing with dismissed merge requests', 'JaCoCo', 'Jenkins', 'Meaningful names for variables, classes, and parameters', 'Naming conventions for feature branches', 'Naming conventions for issues', 'Preferred language in the project', 'SonarLint', and 'SonarQube'. A 'More Pages' button is at the bottom right.

**GitLab** Projects Groups More Search or jump to...

### Knowledge Management for Project UIS (group 11)

**Contents:**

#### 1. Development Guidelines

- [Use of Javadoc for classes and methods](#)
- [Preferred language in the project](#)
- [Common naming patterns for groups of artifacts](#)
- [Meaningful names for variables, classes, and parameters](#)
- [Use of logging levels](#)

#### 2. Handling of Merge Requests and Protected Branches

- [Branching strategy](#)
- [Code review checklist](#)
- [Naming conventions for feature branches](#)
- [Acceptance of merge requests](#)
- [Dealing with dismissed merge requests](#)

#### 3. Quality Improvement

- [Naming conventions for issues](#)
- [Creation of issues](#)

#### 4. Quality Management and Analysis Tool Set-Up in the Project

- [Jenkins](#) (Continuous Integration)
- [SonarQube](#) (Code and Quality Analysis)
- [JaCoCo](#) (Coverage Report)
- [CheckStyle](#) (Static Code Analysis and Coding Styles)
- [SonarLint](#) (Coding Style/Standards Check)

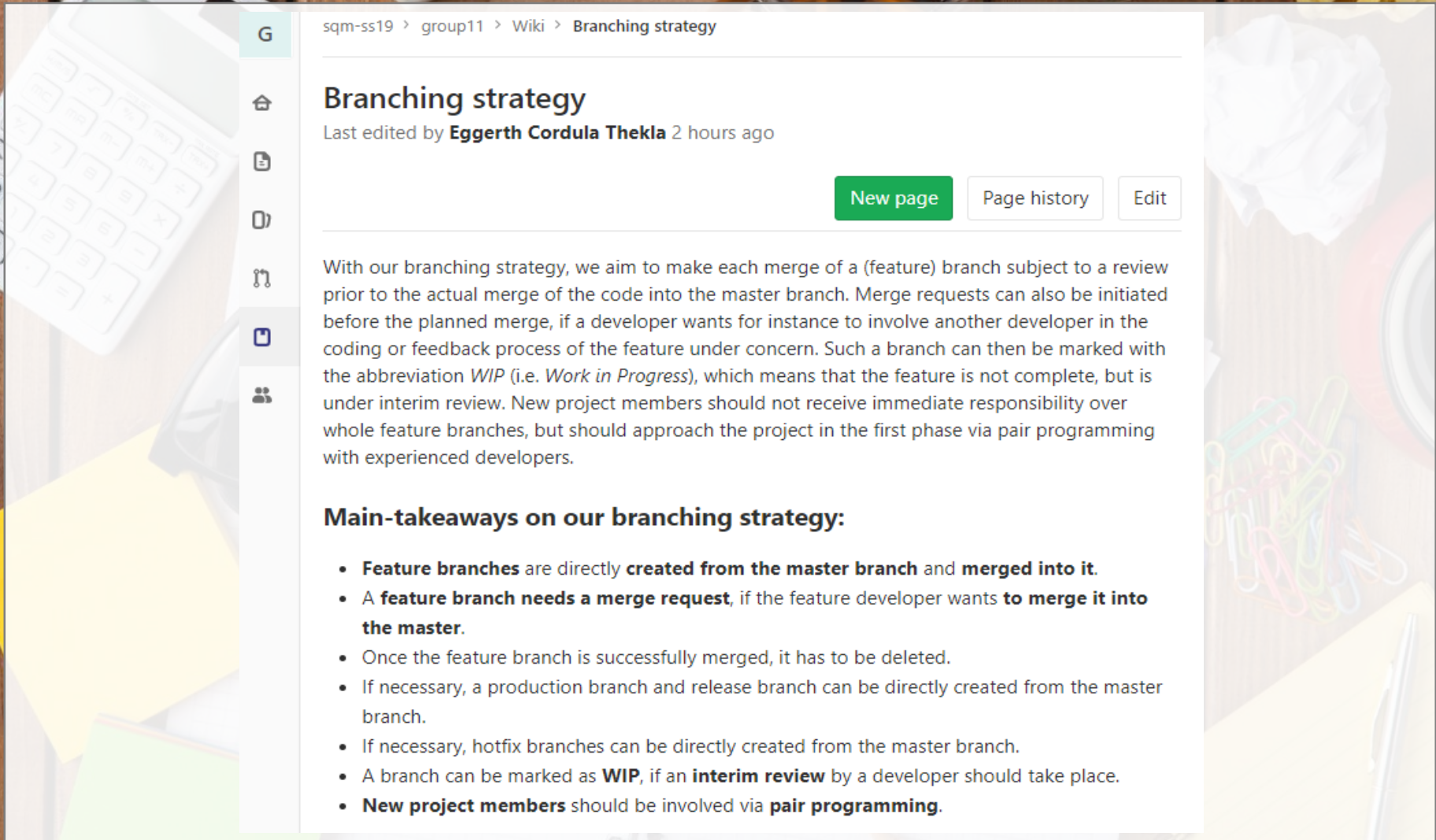
**Clone repository**

- [Acceptance of merge requests](#)
- [Branching strategy](#)
- [CheckStyle](#)
- [Code review checklist](#)
- [Common naming patterns for groups of artifacts](#)
- [Creation of issues](#)
- [Dealing with dismissed merge requests](#)
- [JaCoCo](#)
- [Jenkins](#)
- [Meaningful names for variables, classes, and parameters](#)
- [Naming conventions for feature branches](#)
- [Naming conventions for issues](#)
- [Preferred language in the project](#)
- [SonarLint](#)
- [SonarQube](#)

[More Pages](#)



# Ausschnitt aus dem vorgeschlagenen GitLabWiki (“Branching Strategy“)



The screenshot shows a GitLab Wiki page for a project named 'sqm-ss19'. The breadcrumb trail is 'group11 > Wiki > Branching strategy'. The page title is 'Branching strategy', last edited by 'Eggerth Cordula Thekla' 2 hours ago. There are buttons for 'New page', 'Page history', and 'Edit'. The main content describes the branching strategy, aiming for reviews before merges and using 'WIP' (Work in Progress) branches for features under review. It also lists 'Main-takeaways on our branching strategy' as a bulleted list.

sqm-ss19 > group11 > Wiki > Branching strategy

### Branching strategy

Last edited by **Eggerth Cordula Thekla** 2 hours ago

[New page](#) [Page history](#) [Edit](#)

With our branching strategy, we aim to make each merge of a (feature) branch subject to a review prior to the actual merge of the code into the master branch. Merge requests can also be initiated before the planned merge, if a developer wants for instance to involve another developer in the coding or feedback process of the feature under concern. Such a branch can then be marked with the abbreviation *WIP* (i.e. *Work in Progress*), which means that the feature is not complete, but is under interim review. New project members should not receive immediate responsibility over whole feature branches, but should approach the project in the first phase via pair programming with experienced developers.

#### Main-takeaways on our branching strategy:

- **Feature branches** are directly **created from the master branch** and **merged into it**.
- A **feature branch needs a merge request**, if the feature developer wants **to merge it into the master**.
- Once the feature branch is successfully merged, it has to be deleted.
- If necessary, a production branch and release branch can be directly created from the master branch.
- If necessary, hotfix branches can be directly created from the master branch.
- A branch can be marked as **WIP**, if an **interim review** by a developer should take place.
- **New project members** should be involved via **pair programming**.

# Agenda

1. Vorgeschlagene Qualitätsmanagement-Pipeline
2. Status Quo des Projekts bzgl. Qualität und Verbesserungsvorschläge
3. **Conclusio**

# Conclusio

## Gewonnene Erfahrungen & Einsichten

- Aufbau einer Quality Pipeline durch Verbindung mehrerer Tools
- Verwendung von Code Reviews und Branching-Strategie, um organisiert Wissen zu teilen und Feedback zu geben
- Development Guidelines als strukturierte Hilfestellung für teamorientiertes Arbeiten

## Verbesserungsvorschläge für die Zukunft

- Bedenken des Qualitätsaspekts ganz von Beginn des Softwareprojekts an
- Einführung von agilen Methoden im Projektmanagement (e.g. SCRUM)
- Test-Driven Development (TDD), um bessere Coverage zu erreichen
- Automatisches Deployment auf CI-Server
- Stetige Anpassung und Review der Development Guidelines

Source: <https://www.freepik.com/premium-photo/desk-work-table-brick-wall-background-office-with-laptop>, 207473.htm





**Danke für Ihre Aufmerksamkeit!**

Source: <https://www.ivanti.com/blog/introducing-ivanti-workspace-control-10-2>