Software-Qualitätssicherung im Projekt “Rat um Rad”

Einführung

Dieses Dokument ist ein Leitfaden und Nachschlagewerk für das Qualitätsmanagement- und sicherungsmaßnahmen im Projekt “Rat und Rad”, welche von der Gruppe ProgRadler, bestehend aus Jonah Sebright, Yaowen Rui, Rahel Kempf und Emanuele Tirendi. Qualitätssicherung ist nicht nur für kurzfristige Aspekte wichtig, wie z.B. um den Code lesbarer zu machen und die Fehlerfindung zu vereinfachen. Es hat auch wichtige langfristige Gesichtspunkte, welche von einer hervorragenden Qualitätssicherung profitieren. Code muss Wartungs-Geeignet und die Klassen- und Methoden-Struktur muss zielorientiert und einfach veränderbar geschrieben werden, wobei für letzteres u.A. auch die lines of code per method wichtige Informationen liefern kann. Vor allem ist aber wichtig, dass Qualitätssicherung auch messbar definiert ist, um ihre Ausführung zu gewährleisten.

Merge-Requests sind ein zentraler Punkt in der Qualitätssicherung

Checkliste für Merge-Requests, um eine ausgezeichnete Qualität des Code zu gewährleisten.

- [x] New classes and methods have Javadoc.
- [x] Testable Methods and Classes are tested.
- [x] Project builds (Pipeline).
- [x] All Tests pass (Pipeline).
- [x] Approved and reviewed by a team member.
- [x] Updated documentation documents if needed.
- [x] Code is readable and has no unused code or commented out code.
- [x] Code is properly formatted (Ctrl+Alt+L).

- Die Dokumentation mittels Javadoc erfolgt nach den üblichen Javadoc Konventionen.
- Wir befolgen beim Schreiben unseres Codes Konventionen, welche wir in unserem Team festgehalten haben. Dazu gehören auch einige im Google Java Style Guide (https://google.github.io/styleguide/javaguide.html) festgehaltene Konventionen.

Testing

Es wird angestrebt, alle testbaren Bereiche des Spiels zu testen.
Während der Entwicklung werden Unit-Tests (mit JUnit) geschrieben, um die einzelnen Methoden möglichst durchgehend zu testen. Nach Abschluss einer Teilaufgabe wird die Funktionalität zudem mit manuell durch die Entwickler getestet (Nach Black- und Whitebox Verfahren).

Ein besonderes Augenmerk werden wir im Zuge des im Meilenstein 4 durchzuführenden Testens einer Komponente unseres Spieles auf den Game-Service legen, da dieser als zentraler Baustein des Spiels einen Grossteil der Programmlogik mitbestimmt und prägt und dieser sowohl im Client als auch Server (in Form von GameService und GameServiceUtils) besteht.

Logging

Für das Logging verwenden wir Log4j. Prozesse sollen in separaten Logfiles geloggt werden.
Zum jetzigen Stand gibt es für das ConnectionHandling (Achievement Ping und Pong) ein eigenes Logfile (aktuell als custom Level definiert*). Weitere geloggten Warnungen oder Debugnachrichten werden in einem allgemeinen Logfile geloggt. Zudem unterscheiden wir zwischen den unterschiedlichen Log4J-Levels und geben z.B. Error-Logs zusätzlich in der Commandline aus.

So soll eine gezielte Fehlersuche in unterschiedlichen Bereichen und von unterschiedlichen Personen (Anwender*innen, Entwickler*innen) ermöglicht werden.

* Es ist zu entscheiden, ob ein custom Level auch in Zukunft verwendet werden soll, oder die Umsetzung durch Marker sinnvoller ist.

Teamkultur

Kommunikation mit Respekt, Fehlertoleranz und gemeinsamer Verantwortung.
Förderung der Teamkultur durch Pair Programming (mit IntelliJ “Code with Me”)
Code Reviews zur Qualitätssicherung (z.B. Merge-Requests auf Gitlab reviewen) nach Checkliste aus dem Dokument de06-cs108-FS23-SW_Qualitaetssicherung auf der Seite 36.
Definierte Prozesse (Regelmässige Sitzungen, Zeitplan, Aufgabenteilung → Projektplan, Tagebuch)

Bug Reports: Falls Bugs während des Code Review oder während des Ausprobierens auffallen.

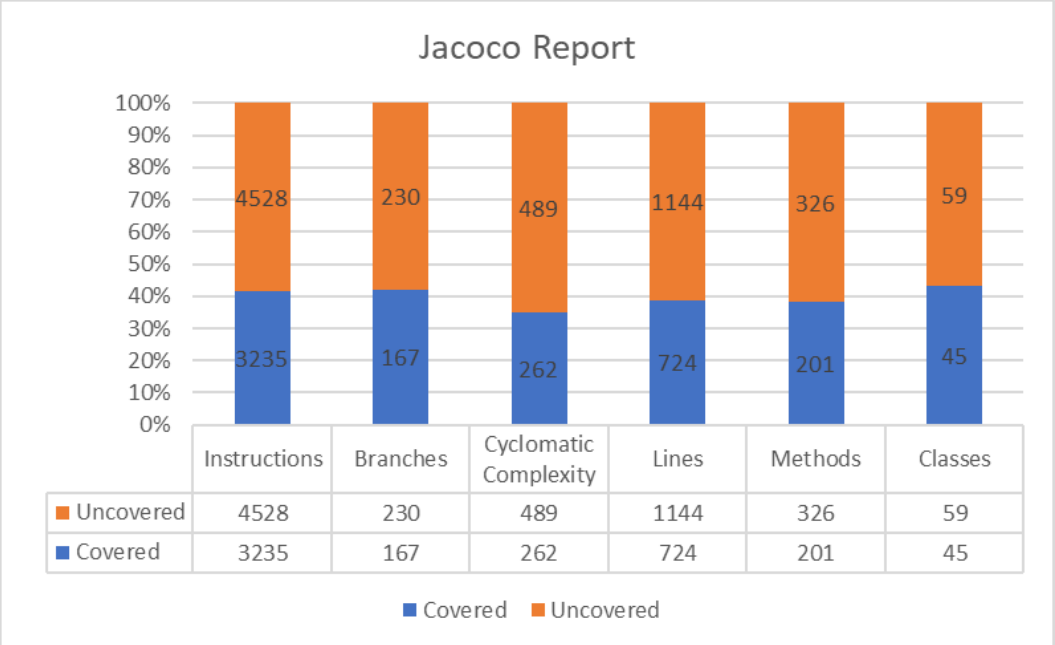
Messungen

Teil unserer Software-Qualitätssicherung sind auch Messungen unseres Repository. Diese umfassen folgende Bereiche:

- Einsatz von Jacoco, um die Code-Coverage zu überprüfen. Für die richtige Interpretation des Jacoco-Reports, siehe: https://www.codementor.io/@etharalali/code-coverage-metrics-cyclomatic-complexity-71n4rrs4r
- Anzahl Logging-Statements im Verhältnis zu Lines of Code
- Metriken: *(pro Metrik Minimum, Maximum und Durchschnitt)*
    1) Lines of Code per Class
    2) Lines of Javadoc per Class
    3) Lines of Code pro per Method

Messungen zum Meilenstein 3: 16.04.2023, 19:00, Änderungen am Code wurden Vorgenommen bis zur Abgabe

Code-Coverage mit Jacoco:



Verhältnis Logging-Statements zu Lines of Code:

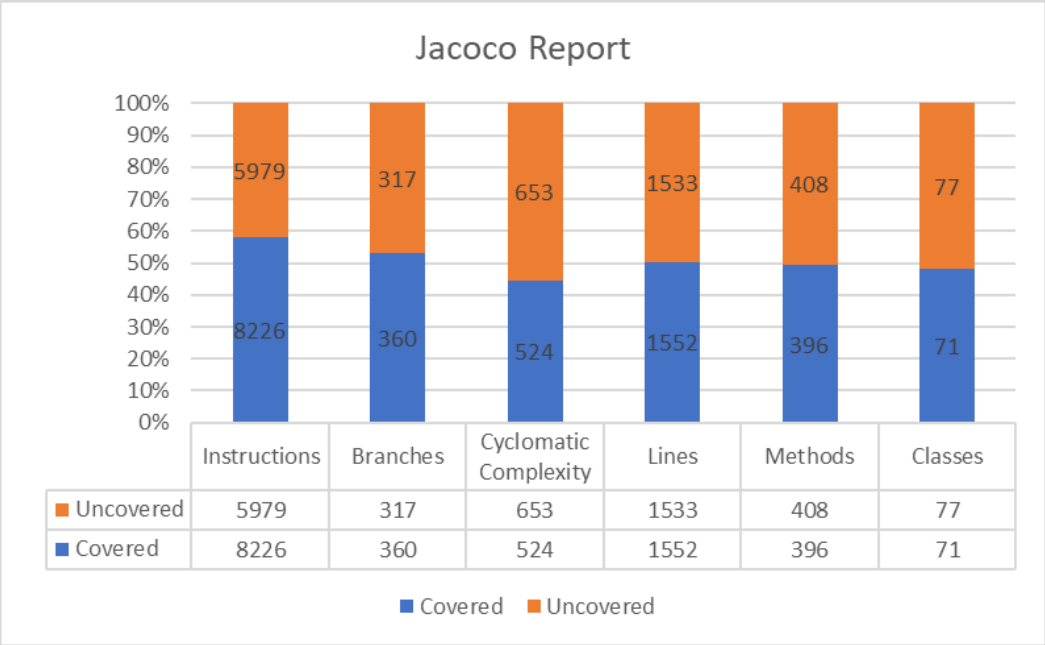
Log-Statements	Lines of Code	Verhältnis
15	5’766	0.26%

Metriken:

	Lines of Code per Class	Lines of Code per Method	Lines of Javadoc per Class
Maximum	284	58	41
Minimum	5	1	0
Average	45.40	8.22	5.38
Total Lines	5’766	4’825	683
Total Classes/Methods	127	587	127

Messungen zum Meilenstein 4: 30.04.2023, 15:00, Änderungen am Code wurden Vorgenommen bis zur Abgabe

Code-Coverage mit Jacoco:



Verhältnis Logging-Statements zu Lines of Code:

Log-Statements	Lines of Code	Verhältnis
36	9’167	0.39%

Metriken:

	Lines of Code per Class	Lines of Code per Method	Lines of Javadoc per Class
Maximum	333	72	41
Minimum	8	1	0
Average	55.22	9.07	5.19
Total Lines	9’167	7’719	861

Total Classes/Methods	166	851	166

Test der wichtigen Kernkomponente: Game-Service:

Dies ist eine Zusammenfassung der drei Klassen: GameService im Client, GameService im Server und GameServiceUtil im Server: