QA-Konzept

Universität Basel FS 22

Programmierprojekt bei Prof. Dr. Heiko Schuldt

Tutorin: Flurina Fischer

Gruppe08 – Night Train to Budapest:

Sebastian Lenzlinger

Jonas Biedermann

Alexandr Sazonov

Seraina Schöb

Inhalt

1 Konzept

Um die Qualität der Software des Projekts sicherzustellen, werden wir uns an folgenden Qualitätsmerkmalen[1] orientieren:

Bedienbarkeit	**Testbarkeit**	**Verständlichkeit**	**Korrektheit**
- Erreichbarkeit	- Strukturiertheit	- Prägnanz	
- Barrierefreiheit	- Abgeschlossenheit	- Lesbarkeit	
- Mitteilungsgüte			

Die Sicherung dieser Merkmale wird in drei Prozessen stattfinden: Beim Coden selbst, in Testungen und in Messungen von Metriken und deren Auswertungen. In den folgenden Abschnitten werden alle drei Prozesse erläutert.

1.1 Coden

Die Qualität des Codes wird bereits beim Coden selbst durch einige Massnahmen sichergestellt. Hier zählen vor allem Lesbarkeit, Prägnanz und Strukturiertheit zu den Fokuspunkten aber auch die Abgeschlossenheit soll gefördert werden. Es werden folgende Massnahmen ergriffen:

(i) Verwendung des *Google styleguides; d*adurch wird die Einheitlichkeit des Codes sichergestellt und somit die Lesbarkeit und Strukturiertheit erhöht.
(ii) *Documentation-oriented-coding*; Es soll in einem ersten Schritt zuerst immer eine Dokumentation erstellt werden, in welcher die angestrebte Funktionalität der Klasse oder Methode beschrieben wird, dann wird implementiert und anschliessend die Dokumentation angepasst und ergänzt
(iii) Mittels Checkliste[2] wird sowohl eine sofortige Selbstprüfung nach der Codingsession als auch eine Prüfung im *Buddy-System* durchgeführt. Dabei wird der Code der Partner:in so schnell als möglich durchgeschaut und eine Rückmeldung gegeben.
(iv) Zweimal wöchentlich - an fixen Sitzungsterminen - wird der Fortschritt und der Code vorgestellt und das weitere Vorgehen besprochen.
(v) Die systematische Verwendung eines Loggers entsprechend den Konventionen.[3]
(vi) Schlussendlich werden schon während des Codingprozesses Ideen für Unittests gesammelt, so dass das Implementieren der Tests leichter fällt.

1.2 Testen

Um die Korrektheit und Bedienbarkeit der Applikation zu gewährleisten, werden Tests durchgeführt. Diese sind in 2 Kategorien eingeteilt:
1) White Box Tests: Mit Unittests wird die Funktionalität der zentralen Bestandteile systematisch überprüft. Damit wird sichergestellt, dass die Teile tun, was sie tun sollen. Die ersten Tests werden gegen Ende des 3. Meilensteins durchgeführt um die ersten Funktionalitäten der Spiellogik zu Prüfen.
2) Black Box Tests: Vor Abgabe jedes Meilensteins wird die Applikation in ihrer Gesamtheit geprüft, um sicherzustellen, dass die Teile im Ganzen tun, was sie sollen. In Meilenstein V wird dieses

[1] Angelehnt an Boehm et al 1976, S. *595*
[2] Vgl. Appendix
[3] Ebd.

Testverfahren vermehrt angewendet werden, um die Erreichbarkeit und Mitteilungsgüte der Benutzeroberflache zu testen und zu verbessern. Falls der Zeitplan es erlaubt, soll in dieser Phase auch die Barrierefreiheit so gut als möglich gewährleistet werden.

1.3 Messen

Um einige der Qualitätsmerkmale quantifizieren zu können, verwenden wir verschiedene Metriken in regelmässigen Abständen (einmal wöchentlich) messen. Die Auswahl der Metriken orientiert sich an den genannten Qualitätszielen, namentlich Testbarkeit und Verständlichkeit.

a) Lines of Code per Class

Diese Metrik besteht aus einer einfachen Zählung von Codelinien im Quellcode. Wir fokussieren uns dabei auf zwei Messungen: (1) *Comment Lines of Code*; hier werden die Kommentarlinien in jeder Klasse gezählt, Leerzeilen werden nicht mitgezählt. (2) *Absolute Lines of Code*; dies zählt die Anzahl Codezeilen inklusive Kommentarzeilen in jeder Klasse ohne Leerzeilen.

Ausgewertet werden die Messergebnisse folgendermassen: Kommentarzeilen werden in ein Verhältnis zur Gesamtzahl der Zeilen gesetzt, was einen Rückschluss auf die Verständlichkeit und Abgeschlossenheit des Programmes zulässt. Weiter ergibt die Differenz der Gesamtzahl der Zeilen und der Kommentarzeilen einen Einblick in die Prägnanz der Codierung. Schlussendlich gibt uns der Unterschied zwischen zwei Messungen in zeitlichem Abstand eine grobe Einsicht in den Fortschritt des Projektes.

b) Zyklomatische Komplexität[4]

Um die Komplexität unseres Programms zu quantifizieren, soll sowohl die Zyklomatische Komplexität nach McCabe für Methoden als auch die *Weighted Methods per Class* nach Chidamber und Kemerer gemessen werden.
Dabei ist die Grundlegende Idee von McCabe, dass die Schwierigkeit eines Programmes von dessen möglichen Flüssen abhängt. Die Metrik hat dabei folgende Bedeutung: Bis 1o «niedrig», bis 20 «mittel» bis 50 «hoch» und ab 50 «undurchschaubar». Je kleiner die Zahl also ist, desto höher ist die Testbarkeit (weniger Fälle) und die Verständlichkeit.

c) Dependants per Class

Diese Metrik misst wie viele andere Klassen von einer Klasse abhängig sind und wie viele Abhängigkeiten jede einzelne Klasse hat. Dies gibt uns einen Einblick in das Abstraktionslevel einzelner Klassen und somit in die Verständlichkeit unseres Quellcodes. Dabei wird dieser Wert immer ins Verhältnis mit der Gesamtzahl aller Klassen gesetzt.

d) Code Coverage

Um die Wirksamkeit unserer Unittests zu prüfen und zu optimieren, wird gemessen, wieviel Code von den Tests abgedeckt wird. Die Abdeckung soll im möglichen Rahmen maximiert werden.

e) Number of Logging-Statements

Diese Metrik soll die Anzahl der Logging-Statements messen und im Verhältnis zu *Lines of Code* gesetzt werden. Dies gibt uns eine Quantifizierung der Verständlichkeit des Quellcodes.

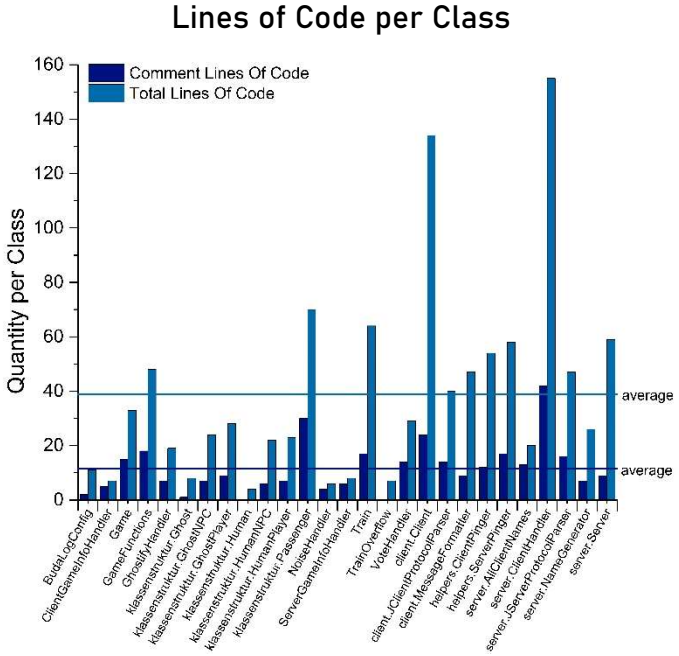
[4] Vgl. Schneider 2021, Kapitel *4.3.2 Zyklomatische Komplexität von McCabe*

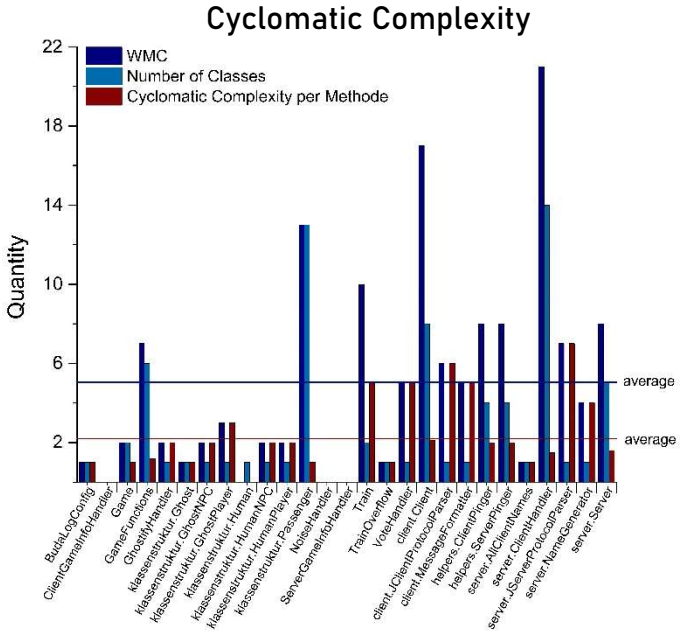
2 Messungen

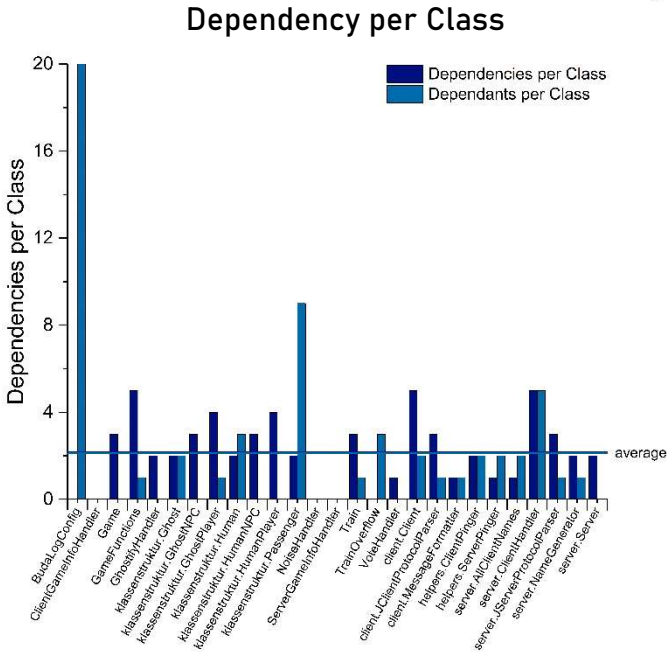
Die Messungen werden mit den JavaPlugins JaCoCo und MetricsReloaded gemessen und anschliessend in ausgewertet.

2.1 Messung 3. April 22

Es wurden die drei Metriken *Lines of Code*, *Cyclomatic Complexity* und *Dependency* gemessen und als Balkendiagramm dargestellt:







Literaturverzeichnis

Schneider K.: *Abenteuer Softwarequalität - Grundlagen und Verfahren für Qualitätssicherung und Qualitätsmanagement,* dpunkt.verlag, Heidelberg, DE: 2012

Boehm B.W.: «Software Engineering» in *IEEE Transactions on Computers*, Vol. 25, Issue 12, pp. 1226 - 1241, Washington DC, USA: 1976

B. W. Boehm, J. R. Brown, and M. Lipow.: «Quantitative evaluation of software quality» in *Proceedings of the 2nd international conference on Software engineering.* IEEE Computer Society Press, pp. 592–605, Washington DC, USA,: 1976

Appendix[5]

Group-Conventions: Log4J Levels meaning:

FATAL	Is fatal for the application (i.e., crashes), needs to be addressed **right now.**
ERROR	Harmful but not fatal, needs to be addressed ASAP, but not everything needs to be stopped right away.
WARN	Needs to get looked at, at some point but is not the highest priority.
INFO	What is the code doing here (i.e., give a return value)
DEBUG	Wherever you would put a *System.out.println()* statement to test your code, instead log with debug.
TRACE	No assigned use so far.

Checklist Code Review:

- Kann ich diesen Code verstehen?
- Hält sich der Code an den *Google styleguide*?
- Ist alles Nötige in der *Dokumentation* vermerkt?
- Ist dieser Code sinnvoll platziert?
- Ist dieser Code redundant?
- Wie lässt sich dieser Code testen?
- Wie kann dieser Code noch verständlicher gemacht werden?
- …

[5] Es wird hier mit Foliensatz 6 *Software-Qualitätssicherung* Folie 65 davon ausgegangen, dass der Appendix mit Checkliste nicht zum Seitenlimit zählt.