# Softwarequalitätssicherungskonzept JDogs

Im Folgenden werden einige Massnahmen beschrieben, wie wir in Zukunft sicherstellen wollen, dass sich unsere Software unseren Vorstellungen entsprechend verhält und so wenig fehlerhaftes Verhalten wie möglich zeigt.

## Konstruktives Qualitätsmanagement

* Wir treffen uns zusätzlich zur Übungsstunde jede Woche am Dienstag um 15:30 Uhr und am Freitag um 10 Uhr, um erledigte sowie anstehende Arbeiten zu besprechen. Wir erscheinen pünktlich zu den vereinbarten Treffen. Bei kurzfristigen Änderungen informieren wir im Gruppenchat auf WhatsApp.
* Wir verwenden ­– wie verlangt – Javadoc, um unseren Code zu dokumentieren. Dabei möchten wir uns an «Design by Contract» halten.
* Exceptions sollten, wenn möglich, sofort behandelt werden, oder es sollte zumindest vermerkt werden, von wem diese Fehlermeldung behandelt werden sollte.
* Wir verwenden einen WhatsApp-Chat, um uns über den Stand der Dinge auf dem Laufenden zu halten und um Probleme mitzuteilen.
* Wir besprechen die Anforderungen, die wir an unser Spiel stellen, detaillierter als bisher und halten diese in Checklisten fest, welche schliesslich auch zur Überprüfung der Software dienen. Damit es zu so wenig Missverständnissen wie möglich kommt, ist es zentral, dass die unterschiedlichen Vorstellungen kommuniziert werden und wir uns auf eine Version einigen, bevor wir mit der Umsetzung von Details beginnen.

## Analytisches Qualitätsmanagement

* Wenn eine Person eine grössere Arbeit erledigt hat, erklärt sie den anderen ihren Code, sodass danach alle mit diesem Code weiterarbeiten können (Structured Walkthrough). Dazu ist es notwendig, dass zuvor die Aufgaben klar verteilt werden und allen klar ist, wer was macht.
* Zusätzlich führen wir Code Reviews durch, wenn eine Komponente fertig gestellt wurde. Dabei stützen wir uns auf die Checkliste aus der Vorlesung (siehe Anhang). Auch dazu ist es wichtig, dass wir im Vornherein festlegen, wann eine Komponente soweit fertig ist, dass sie überprüft werden kann, bevor sie weiterentwickelt wird.
* Um den Black-Box- und den White-Box-Test durchzuführen, helfen die Listen mit den Anforderungen, die zuvor erstellt wurden. Zudem überlegt sich die zuständige Person entsprechende Äquivalenzklassen und erstellt eine Checkliste, die von den anderen Gruppenmitgliedern ggf. ergänzt werden kann. Der Black-Box- und der White-Box-Test sollte jeweils von einer Person durchgeführt werden, die den zu testenden Code nicht geschrieben hat.
* Bugs wollten wir ursprünglich auf Github mittels des Issue Trackers den anderen mitteilen. Stattdessen haben wir aber den WhatsApp-Chat benutzt. Unabhängig davon sollte der gefundene Fehler so genau wie möglich beschrieben werden (Voraussetzungen, Schritte zur Reproduktion, Fehlermeldungen, etc.).
* Wir wollen JUnit-Tests als Hilfsmittel benutzen, um korrekt funktionierenden und wartbaren Code zu schreiben.
* Wir verwenden einen Logger, Log4j2, um besser zu verstehen, wie das Programm abläuft und um auf Fehler aufmerksam zu werden.
* Wir wollen CPU-Usage-Tests einsetzen, um unsere Software zu optimieren.
* Zudem möchten wir Pair Programming (auch zu dritt) bei Gelegenheit einsetzen.
* Die Teammitglieder sollten ihren Code regelmässig einem Quietschentchen o.ä. erklären, um selbst auf allfällige Fehler aufmerksam zu werden.

# Durchführung

Wir hielten uns im Grossen und Ganzen an unser Qualitätssicherungskonzept. Die Messungen führten wir wie verlangt bei den Meilensteinen 3, 4 und 5 durch. Die Resultate der Messungen bei Meilenstein 3 und 4 nahmen wir zum Anlass, unseren Code zu verbessern.

# Resultate

Im Folgenden werden die Resultate der festgelegten Metrics zu den drei Messpunkten präsentiert.

## Javadoc method coverage by class

Diese Metrik gibt an, wie viele Methoden einer Klasse mittels Javadoc dokumentiert sind. Wir entschieden uns für diese Metrik, da ein ausreichend dokumentierter Code die Verständlichkeit desselben erhöht.

|  |  |  |  |
| --- | --- | --- | --- |
| **Meilenstein** | **Min** | **Max** | **Average** |
| 3 | 0% (bei 15 Klassen) | 100% (bei 4 Klassen) | 51.44% |
| 4 | 0% (bei 6 Klassen) | 100% (bei 25 Klassen) | 85.47% |
| 5 | 0% (bei 5 Klassen) | 100% (bei 24 Klassen) | 85.84% |

Tabelle 1: Prozentualer Anteil Methoden einer Klasse, die mittels Javadoc dokumentiert sind.

Diese Entwicklung lässt sich auch anhand der Plots feststellen. Während bei Meilenstein 3 noch bei der Mehrheit der Klassen zwischen 0% und 10% der Methoden mittels Javadoc dokumentiert waren, hat sich das Bild bei Meilenstein 4 und 5 verschoben.

Chart, bar chart

Description automatically generated

Abbildung 1: Javadoc method coverage by class Milestone 3.

Ab Meilenstein 4 sind deutlich mehr Methoden mit Javadoc dokumentiert, da wir uns speziell darauf geachtet haben, den Code ausreichend zu kommentieren. Die Grafik zeigt, dass unsere Bemühungen erfolgreich waren, denn nun sind durchschnittlich über 85% der Methoden einer Klasse dokumentiert. Dass nicht alle Methoden kommentiert sind, liegt unter anderem daran, dass wir es nicht als sinnvoll erachteten, jede toString-, get- und set-Methode zu kommentieren, da diese Methoden mehrheitlich selbsterklärend sind.

Chart, bar chart

Description automatically generated

Abbildung 2: Javadoc method coverage by class Milestone 5.

## Lines of code by class

Diese Metrik gibt die Anzahl Codezeilen pro Klasse an, Wir entschieden uns für diese Metrik, da lange Klassen die Verständlichkeit und Übersichtlichkeit des Codes beeinträchtigen.

|  |  |  |  |
| --- | --- | --- | --- |
| **Meilenstein** | **Min** | **Max** | **Average** |
| 3 | 8 (board.TrackTile) | 781 (ServerGameCommand) | 99.91 |
| 4 | 8 (board.TrackTile) | 916 (RulesCheck) | 132.1 |
| 5 | 8 (board.TrackTile) | 916 (GameWindowController) | 140.81 |

Tabelle 2: Anzahl Codezeilen pro Klasse.

Wie die Tabelle zeigt, stieg die Anzahl Codezeilen pro Klasse von Meilenstein zu Meilenstein. Wie anhand untenstehender Grafik ersichtlich, bilden die extrem langen Klassen jedoch eine Ausnahme. Es gibt nur eine Klasse, die über 900 Codezeilen aufweist und nur zwei mit mehr als 600 Zeilen Code.

Chart, Excel, histogram

Description automatically generated

Abbildung 3: Lines of code by class, Mileston 5.

## Lines of code by method

Diese Metrik gibt die Anzahl Codezeilen pro Methode an, Wir entschieden uns für diese Metrik, da lange Methoden die Verständlichkeit und Übersichtlichkeit des Codes beeinträchtigen.

|  |  |  |  |
| --- | --- | --- | --- |
| **Meilen-**  **stein** | **Min** | **Max** | **Average** |
| 3 | 3  (z.B. Alliance\_4.Alliance\_4(int)) | 162  (ServerMenuCommand.execute(String) | 13.24 |
| 4 | 3  (z.B. Alliance\_4.Alliance\_4(int)) | 205  (ServerMenuCommand.execute(String) | 15.35 |
| 5 | 3  (z.B. Alliance\_4.Alliance\_4(int)) | 212  (ServerMenuCommand.execute(String) | 15.35 |

Tabelle 3: Anzahl Codezeilen pro Methode.

Im Gegensatz zu LOC by class ändert sich die Methode mit den meisten Codezeilen nicht. Die Methode ServerMenuCommand.execute(String) weist zu allen drei Messzeitpunkten am meisten Codezeilen auf. Die Verteilung ist jedoch ungefähr vergleichbar mit LOC by class. Denn auch by LOC by method gibt es nur einzelne Methoden, die äusserst viele Codezeilen aufweisen, wie untenstehende Grafik zeigt.

Chart

Description automatically generated

Abbildung 4: LOC bei Meilenstein 5

## Code Coverage

Obwohl wir uns vorgenommen hatten, den TDD-Ansatz zu verwenden, schrieben wir zuerst den Code und erst danach die Unit-Tests. Erst bei Meilenstein 4 hatten wir sinnvolle Unit-Tests geschrieben, sodass wir lediglich die Code Coverage zu den Messzeitpunkten Meilenstein 4 und Meilenstein 5 betrachten können.

# Diskussion

Wir haben festgestellt, dass es sich lohnt, ein Qualitätssicherungskonzept zu erstellen.

Die Massnahmen, die wir in Kapitel 1 definiert haben, konnten wir jedoch nicht alle einhalten. Wir haben uns wie vereinbart getroffen und uns über WhatsApp auf dem Laufenden gehalten. Bei Meilenstein 3 stellten wir fest, dass wir unsere Arbeitsweise ändern müssen, wir wollten die Aufgaben klarer definieren und fixe Deadlines vereinbaren.

Aufgrund der Messungen wussten wir, wo wir uns verbessern konnten. So erreichten wir bei Meilenstein 4 beispielsweise einen deutlich höheren Anteil an Methoden pro Klasse, die mittels Javadoc dokumentiert waren.

Wir stellten fest, dass Klassen und Methoden schnell sehr lang werden. Dies ist wohl nicht sehr übersichtlich, jedoch fragten wir uns, ob viele kurze Klassen und Methoden den Code verständlicher machen, da dann jeweils zwischen den Klassen und Methoden hin- und hergesprungen werden muss.

Neben den eher technischen Qualitätssicherungsmassnahmen ist jedoch eine funktionierende Kommunikation äusserst wichtig. Wenn nicht klar ist, was bis wann wie erreicht werden soll, helfen Tests und Messungen nicht weiter. Deshalb waren für unser Projekt die regelmässigen Treffen und die Kommunikation über WhatsApp entscheidend.

# Anhang

## Checkliste Code Review

1. Verstehe ich diesen Code?
2. Hält sich der Code an den vereinbarten Programmierstil?
3. Würde ich diesen Code an dieser Stelle im Projekt erwarten?
4. Hat es bereits Code mit ähnlicher Funktionalität an anderer Stelle?
5. Lässt sich die Lesbarkeit dieses Codes erhöhen?
6. Könne ich diesen Code warten und Änderungen vornehmen?
7. Ist der Code ausreichend dokumentiert?

## Checkliste Milestone2

1. Verbindung Client-Server
   1. Richtige Daten (IP-Adresse, Portnummer) eingeben.
   2. Falsche Daten (IP-Adresse, Portnummer) eingeben.
   3. Client will sich mit Server verbinden, aber der Server ist noch nicht gestartet.
   4. Client ist mit Server verbunden, Server wird abgeschaltet.
2. Login
   1. Dem Client wird ein Username vorgeschlagen, basierend auf dem Systemnamen
   2. Client wählt bereits vergebenen Namen.àNeuer Name wird vorgeschlagen.
   3. Client kann seinen Namen ändern.
   4. Client kann sich ausloggen.
3. Chat
   1. Nachricht wird allen aktiven Clients angezeigt.
   2. Es ist ersichtlich, wer die Nachricht geschrieben hat.
   3. Protokollbefehle werden entsprechend behandelt.
   4. Sonderzeichen in Nachrichten werden korrekt angezeigt.
   5. Emojis in Nachrichten werden korrekt angezeigt.

## Checkliste Milestone3

1. Lobby
   1. Spielstatistik kann aufgerufen werden.
   2. Andere Spieler werden angezeigt.
   3. Public Chat funktioniert in der Public Lobby.
   4. Public Chat funktioniert, in und von der GameLobby.
   5. WhisperChat funktioniert, wenn sich der Absender und Empfänger in der Public Lobby befinden.
   6. WhisperChat funktioniert nicht, wenn sich der Absender oder Empfänger in der GameLobby befinden.
2. Spiel starten
   1. Ein Spiel eröffnen mit OGAM.
   2. Einem Spiel beitreten mit JOIN.
   3. Das Spiel startet nur, wenn genügend Spieler beigetreten sind.
   4. Nur der Host kann das Spiel starten mit STAR.
3. Spielen (Allgemein)
   1. Die Reihenfolge der Spieler wird eingehalten.
   2. Es wird ein zufälliger Beginner ermittelt.
   3. Ein Spieler kann nur spielen, wenn er am Zug ist.
   4. Die Spieler erhalten entsprechende Anzahl Karten.
   5. Spieler erhalten vier Murmeln auf HomeTiles.
   6. Vom Ende des Spielbretts gelangt man wieder an den Anfang.
   7. Wenn die Spieler keine Karten mehr haben, erhalten sie neue.
4. Spielzug
   1. Spieler kann gewünschte Karte aus seiner Hand wählen.
   2. Wenn eine Karte gespielt wird, verschwindet sie von der Hand des Spielers.
   3. Spieler kann nur Aktionen durchführen, die mit der gespielten Karte erlaubt sind.
   4. Ein Spieler kann nur seine Murmeln bewegen (richtige Farbe).
   5. Wenn ein Spieler auf ein besetztes Feld gelangt, wird die Murmel, die vorher auf dem Feld war, in den Zwinger geschickt.
5. Spiel beenden
   1. Der Gewinner wird korrekt ermittelt
   2. Spieler gelangen zurück in die Lobby
   3. Das Spiel kann vorzeitig mit QUIT oder EXIT beendet werden

## Checkliste Milestone4

1. Lobby
   1. Spielstatistik kann aufgerufen werden.
   2. Andere Spieler werden angezeigt.
   3. Public Chat funktioniert in der Public Lobby.
   4. Public Chat funktioniert, in und von der GameLobby.
   5. WhisperChat funktioniert, wenn sich der Absender und Empfänger in der Public Lobby befinden.
   6. WhisperChat funktioniert nicht, wenn sich der Absender oder Empfänger in der GameLobby befinden.
2. Spiel starten
   1. Ein Spiel eröffnen mit Button Start
   2. Einem Spiel beitreten mit Button JOIN.
   3. Das Spiel startet nur, wenn genügend Spieler beigetreten sind.
   4. Nur der Host kann das Spiel starten mit dem Start-Button.
3. Spielen (Allgemein)
   1. Die Reihenfolge der Spieler wird eingehalten.
   2. Es wird ein zufälliger Beginner ermittelt.
   3. Ein Spieler kann nur spielen, wenn er am Zug ist.
   4. Die Spieler erhalten entsprechende Anzahl Karten.
   5. Spieler erhalten vier Murmeln auf HomeTiles.
   6. Vom Ende des Spielbretts gelangt man wieder an den Anfang.
   7. Wenn die Spieler keine Karten mehr haben, erhalten sie neue.
4. Spielzug
   1. Spieler kann gewünschte Karte aus seiner Hand wählen.
   2. Wenn eine Karte gespielt wird, verschwindet sie von der Hand des Spielers.
   3. Spieler kann nur Aktionen durchführen, die mit der gespielten Karte erlaubt sind.
   4. Ein Spieler kann nur seine Murmeln bewegen (richtige Farbe).
   5. Wenn ein Spieler auf ein besetztes Feld gelangt, wird die Murmel, die vorher auf dem Feld war, in den Zwinger geschickt.
   6. Jack funktioniert korrekt (Murmeln auf Tracktile tauschen).
   7. Spezialkarte 7 funktioniert korrekt:
      1. Alle von der 7 überholten Murmeln werden nach Hause geschickt.
      2. 7 kann auf mehrere eigene Murmeln aufgeteilt werden.
      3. 7 kann auf eigene und Murmeln des Partners aufgeteilt werden.
   8. Nur mit der Karte 4 kann man rückwärts laufen.
   9. Eine Murmel, die erst gerade aus dem Zwinger gekommen ist, blockiert den Durchgang für die anderen Murmeln.
   10. Eine Murmel, die erst gerade aus dem Zwinger gekommen ist, darf nicht direkt ins Ziel laufen.
   11. TeamMode: Wenn alle Murmeln eines Spielers im Ziel sind, darf dieser Spieler mit den Murmeln seines Teampartners fahren.
5. Spiel beenden
   1. SingleMode: Der Gewinner wird korrekt ermittelt
   2. TeamMode: Das Gewinnerteam wird korrekt ermittelt
   3. Spieler gelangen zurück in die Lobby
   4. Das Spiel kann vorzeitig mit beendet werden.
      1. Zurück in die Lobby gehen
      2. Das Spiel komplett verlassen (Verbindung zum Server trennen)