Auftragnehmer:
Projektgruppe 06 “Green Twelve Studios“
Ansprechpartnerin: Mirja Kühn
E-Mail: mkuehn@mail.upb.de

QA-Dokument

Definition of Done

Eine User-Story und der dazugehörige Quellcode sind als “done“ einzustufen, wenn die folgenden Punkte zufriedenstellend umgesetzt sind:

a) User-Stories

- Die User-Stories wurden zur Zufriedenheit des Teams umgesetzt

b) Code Conventions

- Der Code muss den Standards des Google Java Style Guides entsprechen

c) Dokumentation

- Der Quellcode ist gemäß der in b) beschriebenen Code Conventions dokumentiert
- Der Quellcode wurde anhand von UML-Diagrammen geschrieben und ggf. wurden entsprechende UML-Diagramme aktualisiert

d) Test

- Alle Tests wurden wie im Testplan beschrieben erfolgreich abgeschlossen
- Alle Veränderungen/Verbesserungen, die sich durch vorheriges Testen ergeben haben sind vorgenommen und selbst getestet

e) Abnahme

- Alle Akzeptanzkriterien werden erfüllt
- Der Team hat die User-Story akzeptiert
- Alle Implementierung des Features wurden dem Develop-Branch hinzugefügt

Erläuterung

a) User-Stories

Die User-Stories erhalten die Prioritäten von 1 bis 3, wobei 3 die höchste Priorität darstellt. Die Abarbeitung erfolgt gemäß der Priorität, sofern nicht andere Aufgaben kurzfristig als dringlicher eingestuft werden.
Es wird zwischen dem Entwicklungsteam und dem Product Owner abgestimmt, wann eine User-Story fertig ist. Dadurch, dass klare Erwartungen an die Software existieren und eine allgemeingültige Checkliste besteht, wird so qualitativ und schnell gearbeitet. Um dies zu gewährleisten wird für jede User-Story eine Menge an Akzeptanzkriterien - dargestellt als Sub Tasks - erstellt, wodurch die Fertigstellung der User-Stories mittels Softwaretests überprüft werden kann. Auch helfen die Akzeptanzkriterien, jedem im Team ein Verständnis der User-Stories zu vermitteln.

b) Code Conventions

Damit der Code eine hohe Qualität besitzt und später gut wartbar, bzw. erweiterbar ist, soll er den Standards entsprechend, die der Google Java Style Guide vorgibt.

Die wichtigsten Conventions werden im Folgenden erläutert:

Benennung:
Es ist besonders auf die Wahl von aussagekräftigen Namen zu achten. Es wird im Allgemeinen die CamelCase-Schreibweise genutzt. Die Benennung erfolgt in Englischer Sprache. Im Gegensatz dazu werden Konstanten im Constant Case benannt.

Einrückung von Blöcken:
Jeder Block soll im Vergleich zur vorherigen Ebene zur besseren Übersichtlichkeit eingerückt sein. Hierauf ist zum Beispiel bei if-Bedingungen, Schleifen und nach dem Beginn von Methoden und Klassen zu achten. Außerdem sollen bei if-Bedingungen und Schleifen immer geschweifte Klammern genutzt werden, auch wenn der dadurch entstehende Block nur eine oder gar keine Anweisung enthält.

Dokumentation:
Die Dokumentation der Methoden soll dem Javadocs Standard entsprechen, hierbei muss zu allen nicht trivialen Methoden (Getter/ Setter sind trivial), eine kurze Beschreibung der Funktion, sowie eine kurze Erklärung zu den der Funktion übergebenen Parameter und dem Rückgabewert aufgeführt sein. Außerdem sollen die Exceptiones angegeben werden, welche von der Methode geworfen werden können.

Zusätzlich soll der Quellcode wo es nötig ist mit zusätzlichen Kommentaren versehen werden.

Beispiel zu Code Conventions:

```
double EULERS_NUMBER = 2.71828182846; //Constants in constant case
/** This function calculates the faculty of the given parameter n
*
* @param n
* @return the faculty of n
* @throws ArithmeticException if n is negative
*/
public static int calculateFaculty(int n){ // function name in CamelCase
if(n<0){//parenthesis, although there is just one instruction
throw new ArithmeticException("The factorial of a negative number is not defined");
} else if (n==0 || n==1){
return 1; //new line after the parethesis
} else {
return n*calculateFaculty(n-1);//Recursive call of the function
}
}
```

c) Dokumentation

Der in dem jeweiligen Sprint geschriebene Quellcode wurde nach den in b) beschriebenen Standards für JavaDoc auf Englisch dokumentiert.

Die REST-APIs unseres Software-Paketes sind mittels geeigneter UML-Klassendiagramme, UML-Sequenzdiagramme sowie UML-State-Charts in unserem Interface-Dokument im Voraus zu dokumentieren und anhand dieser zu programmieren.
Das Interface-Dokument ist bis zum 07.12.2020 fertigzustellen.

Alle anderen Komponenten unseres Software-Paketes sind ebenfalls mittels geeigneter UML-Klassendiagramme sowie UML-Aktivitätsdiagrammen zu dokumentieren und anhand dieser zu programmieren. Diese UML-Diagramme werden allerdings zusammen mit einer kurzen Zusammenfassung der einzelnen Komponenten in Bezug auf folgende Leitfragen in das DevOps-Dokument integriert:

- Welche Features wurden aus welchem Grund in der Komponente implementiert?
- Welche Libraries, Frameworks oder andere Technologien wurden mit welchem Ziel verwendet?
- Welche Abhängigkeiten in Bezug auf die REST-API der anderen Komponenten besteht zur dieser Komponente?

Das DevOps-Dokument ist zusammen mit der Implementierung der Messe Version bis zum 08.01.2021 fertigzustellen.

Für die Endabgabe erstellen wir außerdem README's zu jeder Komponente auf unserem GIT.

Das gesamte Softwarepaket ist allerdings erst bis zum 05.02.2021 fertigzustellen.

d) Test

Für die Tests gilt die im Testplan festgelegte Vorgehensweise.
Sobald alle zu den relevanten Fällen gehörenden Tests erfolgreich abgeschlossen wurden, kann die User-Story als "done" eingestuft werden. Als relevante Fälle bezeichnen wir diese, die unerlässlich für die Funktionalität der Software sind.

e) Abnahme

Sobald das Feature alle beschriebenen Anforderungskriterien erfüllt und gemäß den anderen Anforderungen dieser Definition of Done als "done" eingestuft wurde, kann dieses der lauffähigen Produktversion im Develop-Branch hinzugefügt werden. Sollte das Feature noch nicht alle Anforderungen erfüllen, muss das Team notwendige Anpassungen besprechen und es wird dementsprechend im nächsten Sprint erneut bearbeitet und erst dann dem Develop-Branch hinzugefügt, sofern es im nächsten Sprint abgenommen wurde.

Testplan

Jede in einem Sprint entwickelte/überarbeitete Softwarekomponente muss die folgenden Tests durchlaufen und erfolgreich abschließen, bevor sie dem auszuliefernden Produkt hinzugefügt werden kann.

a) Modul-Test
b) Code-Review
c) Integrationstest
d) Systemtest
e) Performance-Test
f) Akzeptanztest

Damit ein entwickeltes Feature als "done" gemäß der Definition of Done eingestuft werden kann, müssen mindestens die Test a) bis c) und gegebenenfalls d), wenn es sich um eine Komponente übergreifende User- Story handelt, erfolgreich durchgeführt worden sein.

Erläuterung

a) Modul-Test

Unsere Modultests sollen die Funktionalität einzelner Code-Abschnitte und User-Stories sicherstellen. Mit Hilfe von manuellen Tests werden einzelne kleinere Code-Abschnitte wie z.B. Klassenmethoden getestet. Dabei wird sichergestellt, ob der Code den Anforderungen der User-Stories entspricht. Es wird außerdem auf die Unabhängigkeit der Tests, dem Testen von Grenzfällen als auch der Dokumentation der Tests geachtet. Für umfangreichere Code-Abschnitte, in denen ein einfacher Modultest nicht ausreicht, werden wir automatisierte Tests mit JUnit erstellen. Dabei werden sowohl Blackbox-Tests als auch Whitebox-Tests erstellt. Durch Regressionstests überprüfen wir zudem stets die Funktionalität bei Änderung des Codes.

b) Code-Review

Für jede zu entwickelnde Komponente wird eine Code-Review durchgeführt. Dies bedeutet, dass ein anderes Mitglied des Projektteams den Code hinsichtlich logischer Fehler und Einhaltung der Code Conventions prüft. Außerdem sollen sich der Entwickler des Codes und der Prüfer gegebenenfalls über mögliche Verbesserungen bezüglich der Laufzeit austauschen. Dadurch soll zum einen überprüft werden, ob nicht noch eine für die Laufzeit bessere Methode der Implementierung gefunden werden kann und zum anderen wird durch die Prüfung der Code Convention gleichzeitig die Qualität der Dokumentation gewährleistet.

c) Integrationstest

Integrationstests decken die Funktionstest mehrerer voneinander abhängigen, allerdings für sich funktionierenden Teile des Codes ab, wenn diese zusammengeführt werden. Wir werden sie immer dann einsetzen, wenn wir nach / während erfolgreichen Sprints wieder neue Teile des Programms fertigstellen konnten und nun die Funktionalität im Ganzen getestet werden muss. Beispielsweise funktioniert die KI alleine einwandfrei und der Spieleserver auch. Wenn nun allerdings die KI einen bestimmten Zug auf dem Spielserver machen will tritt ein Fehler auf. Einen solchen Fehler im Zusammenspiel mehrerer Komponenten, können wir durch einen Integrationstest frühzeitig erkennen.

d) Systemtest

Systemtests simulieren unter möglichst realistischen Bedingungen die Anwendung des bereits fertig integrierten Systems. Sie stehen also in der Testhierarchie über den Integrationstests, die hauptsächlich während der Entwicklung durchgeführt werden. Andererseits beziehen sich die Systemtests eher auf ein auslieferungsfähiges Programm (oder System) und dort generelle Anwendungsprobleme sicherstellen. Wir werden Systemtests vorrangig vor jeder Abgabe mit dem derzeitigen Stand unseres Programms durchführen. Davon versprechen wir uns, letzte Fehler des Gesamtsystems ausfindig zu machen, welche in der tatsächlichen Anwendung auffallen.

e) Performance-Test

Einerseits wird der Perfomance-Test unserer Software durch die Usability-Tests abgedeckt, andererseits sind bestimmte Vorgaben bezüglich der Performance wie z.B. die Entscheidungsgeschwindigkeit der KI gegeben, die wir durch separate Tests überprüfen.

f) Akzeptanztest

Nachdem das Softwareprodukt die oben genannten Tests erfolgreich bestanden hat, wird der Akzeptanztest als letzter Test durch Product-Owner und Testmanager unter den tatsächlichen Nutzungsbedingungen durchgeführt. Mithilfe von Black-Box-Testing testet der Akzeptanztest das Gesamtsystem, insbesondere die Kerngeschäftsprozesse, um die Einsatzbereitschaft des Softwareprodukts sicherzustellen. Im Vergleich zwischen Testergebnis und dem vorgesehenen Standard wird die Entscheidung getroffen, ob das fertige Produkt akzeptierbar ist.

Verwendete Tools:

Wir nutzen JUnit um die einzelnen Komponenten zu testen. Diese Tests werden automatisiert, aber auch manuell mit Hilfe von Maven ausgeführt. Außerdem werden bei jedem Commit alle Tests zu den jeweiligen Komponenten mit Hilfe von Gitlab CI ausgeführt, sodass ein Merge von der Feature-Branch auf die Develop-Branch nur beim Bestehen von allen Tests möglich ist. Durch das automatische Testen bei jedem Merge, verringert sich die Wahrscheinlichkeit, dass fehlerhafter oder unfertiger Code in die fertige Komponente übernommen wird. Für eine bessere Code Coverage nutzen wir weiterhin die Open-Source Library JaCoCo.