Michel Romancuk, Lukas Schmidt, Yash Trivedi, Severin Memmishofer

March 2023

Quality Assurance Concept Gruppe 7 - Key H(a)unt

Ziel des QA Konzepts

Um die Qualität des Endproduktes sicherzustellen, soll die Entwicklung von Key H(a)unt durch ein "Quality Assurance Concept" garantiert werden. Mit dem QA Concept wird sichergestellt, dass:

(i) Run-time Analyse und Debugging des Codes einfach ist.

(ii) Der Code sich in wenigen Schritten testen lässt.

(iii) Der Code verständlich und gut dokumentiert ist.

(iv) Änderungen im Code zielgerichtet an die Anforderungen sind.

Welche Massnahmen werden ergriffen, um diese Ziele zu erreichen?

(i) Run-time Analyse und Debugging des Codes

Um das Debugging zu erleichtern, werden wir einen Logger verwenden. Dieser soll folgende Funktionen unterstützen: Verschiedene Log-level (und entsprechende Ausgabe, je nach Fehlertyp), Generierung von Log-Files mit Zeitangabe und Ausgabe in Echtzeit. Dieser sollte auch komplett ausschaltbar sein, um Runtime- und Performance-Analyse durchzuführen, falls gewünscht. Um genügend Verbosität zu erreichen, wird die relative Anzahl Logging-Statements überprüft.

(ii) Testbarkeit des Codes

Es müssen zu kritischen Stellen unseres Codes $Unit - Tests$ erstellt werden. Die $Unit - Tests$ müssen erfolgreich abgeschlossen werden, bevor z.B ein Merge unternommen werden kann. Um sicherzustellen, dass es genug $Unit - tests$ hat, wird die Code-Coverage regelmässig überprüft.

(iii) Verständlichkeit und Dokumentation des Codes

Die Nutzung von Javadoc wird garantieren, dass der Code verständlich und gut dokumentiert ist. Die Verwendung eines einheitlichen Programmierstils wird mittels Checkstyle, in Anlehnung an die Sun Coding Conventions, durchgersetzt. Cyclomatic complexity, relative Anzahl an Zeilen Javadoc und Zeilen Code pro Klasse werden gemessen um Code-Komplexität niedrig zu halten und Erweiterbarkeit zu vereinfachen.

(iv) Erfüllung der Anforderungen

Mittels gruppeninternem Code-Review wird schlussendlich garantiert, dass der Code alle obigen Ziele erfüllt und dass neue Features sowohl den vorgegebenen Meilensteinen, als auch den projektbezogenen Anforderungen an unser Produkt entsprechen.

Messungen

Die Messungen werden mit dem MetricsReloaded gemessen und anschliessend ausgewertet.

Erste Messungen 15. April 2023

Es wurden drei Metriken sinnvoll gemessen: Lines of Code, Cyclomatic Complexity und Dependency sowie Logs pro Klasse. Es wurden nur ein Unit Tests durchegführt. Dieser wird nicht aufgezählt

In folgender Tabelle sieht man, wie gut die einzelnen Klassen mit Javadoc abgedeckt sind. Insgesamt gibt es momentan 929 Zeilen von Javadoc und somit gibt es im durchschnitt 116 Zeilen von Javadoc pro Klasse. Schlussendlich sind ganze 92.31%. Wir haben 2004 Zeilen normalen Java-code. Inzgesamt sind es bis zum 15.April 2933 Zeilen Code. Man sieht, dass die "Javadoc fieldcoverage" das schwächste glied ist. Dies liegt daran, dass wir bei Feldern klare und eindeutige Namen benutzen.

Package	Javadoc lines of code	Javadoc Class coverage	Javadoc fieldcoverage	Javadoc methode coverage
client	82	75.00%	0.00%	91.67%
client.gui	86	75.00%	22.73%	100.00%
net	183	75.00%	71.43%	94.44%
player	32	100.00%	0.00%	100.00%
server	217	83.33%	25.00%	94.55%
server.game	109	75.00%	22.22%	91.67%
server.game.utility	217	75.00%	32.26%	85.42%

In der folgenden Tabelle sieht man die Code Dependencies. Was einem direkt auffällt, ist dass das Server-Packet sowie das net-Packet die meisten Abhängikeiten haben. Das net-Packet had die meisten transitiven Abhängikeiten. Das liegt daran, dass das Netzwerkprotokol vom Klienten sowie auch Server genutzt wird.

Package	Cyclic	Number of Package dependencies	Number of dependent packages	Number of transitively dependent packages
client	1	2	2	3
client.gui	1	2	1	3
net	0	0	7	8
player	3	2	3	5
server	3	4	3	5
server.game	3	4	1	5
server.game.utility	3	2	2	5

In der folgenden Tabelle sieht man die Cyclomatic complexity. Kein Packet hat eine kritische Komplexität erreicht. Einzelne Klassen sind jedoch im kritischen bereich. Die Main-Klasse ist fundamental kritisch. Da die Spiel-Logik beim Server verarbeitet wird, ist die Spiel-Logik Klasse auch kritisch.

Package	Average cyclic complexity	total cyclic complexity
client	2.79	67
client.gui	1.50	24
net	1.82	31
player	1.00	9
server	2.22	122
server.game	2.54	61
server.game.utility	1.49	67

Zum Stichtag gibt es 28 Logger-statements und sind wie folgend verteilt: 7 trace, 13 info, 1 warn, 4 error, 3 debug.