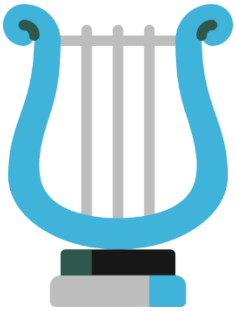


Testkonzept

Vorgehensweise, Mittel und Ablaufplan der Testaktivitäten

Version 4.0

Team Apollon:

Paul-Benedict Burkard

Florian Albert

Leon Jerke

Daniel Kröker

Alfred Rustemi

Etienne Zink

Projektbezeichnung	APOLLON
Projektleiter	Paul-Benedict Burkard
Verantwortlich	Leon Jerke
Versionsdatum	18.04.2021

Inhaltsverzeichnis

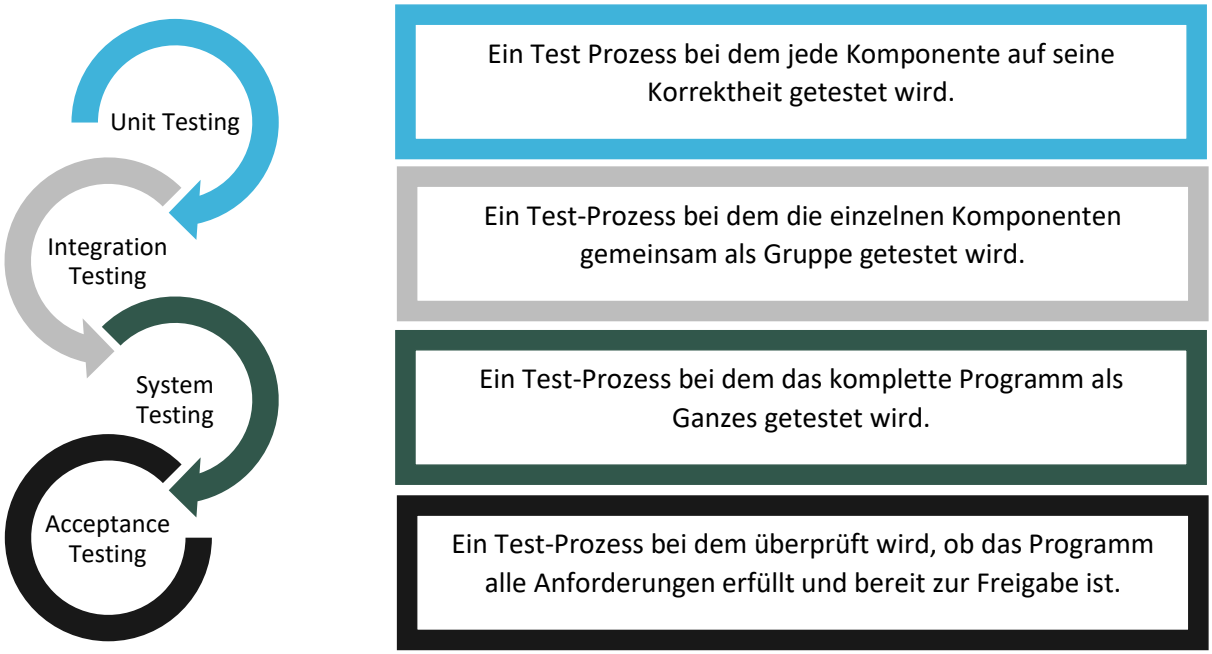
1. Einführung

Ziel des Projekts ist die Entwicklung einer Software für einen Kunden. Der Kunde soll den Server übernehmen und selbstständig betreiben. Durch die Software sollen Spieler über den Web Browser textbasierte Multi User Dungeons selbst konfigurieren und an von anderen Spielern konfigurierten Dungeons, mit einem erstellten Avatar, teilnehmen können.

Für die Entwicklung von Software ist das Testen dieser essenziell, um die Funktionalität zu bewerten. Testing dient dazu festzustellen, ob die entwickelte Software die spezifizierten Anforderungen erfüllt oder nicht. Durch Tests werden Mängel identifiziert und sichergestellt, dass das Produkt einwandfrei funktioniert. Allgemein sparen Tests grundsätzlich Geld, bieten mehr Sicherheit, eine bessere Produktqualität und eine höhere Kundenzufriedenheit Im folgenden Bericht wird das geplante Testkonzept unseres Produkts genauer erläutert.

1.1 Teststufen

Das Testkonzept unterteilt sich in vier wesentliche Teststufen:



1.1.1 Definition von Modul-/Komponententests (engl. Unit Tests)

Modultests werden beim Programmieren eingesetzt, um individuell einzelne Module des Codes zu testen bzw. sie auf ihre Korrektheit zu überprüfen. Ein Modul beschreibt dabei eine Einheit eines Programms. In den meisten Fällen entspricht so eine Einheit einer Klasse, kann jedoch auch aus

mehreren Klassen bestehen. Modultests sind die erste Stufe von Softwaretests und werden vom Entwickler selbst erstellt. Die Durchführung dieser Tests erfolgt automatisiert.

1.1.2 Definition von Integrationstests

Modultests gewährleisten lediglich, dass die Komponenten individuell funktionieren. Integrationstests dagegen prüfen mehrere Komponenten gemeinsam. Sie sind eine aufeinander abgestimmte Reihe von Einzeltests, die überprüfen, ob die Kommunikation mit anderen Modulen reibungslos funktioniert. Integrationstests bilden die zweite Stufe der Softwaretests.

1.1.3 Definition von Systemtests

Für den Systemtest wird das komplette Programm kompiliert und als Ganzes getestet. Hierbei gibt es drei verschiedene Arten, auf die es getestet werden kann:

- Funktionstest: Testet, ob das Programm so funktioniert wie es soll.
- Leistungstest: Testet, wie effizient das Programm in verschiedenen Szenarien ist.
- Portabilitätstest: Testet, wie das Programm auf verschiedenen Plattformen arbeitet.

Systemtests bilden die dritte Stufe der Softwaretests.

1.1.4 Definition von Akzeptanztests

Wenn das Programm bereit ist zur Übergabe an den Kunden, muss es zunächst nochmal grundlegend getestet werden. Hierbei gibt es zwei Phasen:

- Alpha-Test: Die Entwickler selbst testen das Programm so wie es in seinem geplanten Arbeitsumfeld genutzt werden soll.
- Beta-Test: Das Programm wird an die zukünftigen Nutzer übergeben, um es zu testen.

Akzeptanztests bilden die vierte und letzte Stufe der Softwaretests.

2. Testplan

2.1 Modul-/Komponententests (engl. Unit Tests)

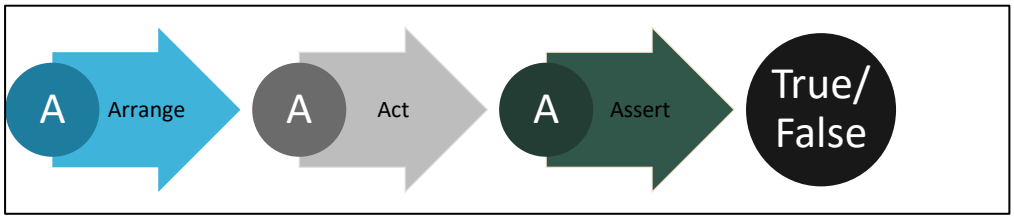
Komponententests werden direkt von Beginn an für alle Klassen, bei denen es möglich ist, geschrieben. Die Implementierung erfolgt hierbei in der Regel von derselben Person, die den Code geschrieben hat. Ziel ist es jede geschriebene Methode mit mehreren Tests auf verschiedene Fälle zu überprüfen, um

so eine ausreichende Fehlerfreiheit zu gewährleisten. Der Inhalt der Tests muss verständlich und übersichtlich aufgebaut sein.

Wie werden die Tests erstellt?

Das Erstellen der Tests erfolgt mit dem Testframework xUnit. Zusätzlich zum xUnit-Framework wird die Mocking-Library NSubstitute in Kombination mit AutoFixture verwendet. NSubstitute ermöglicht das Isolieren von einzelnen Komponenten und AutoFixture generiert die benötigten Testdaten. Auf diese Weise können Komponententests schneller und effizienter erstellt werden.

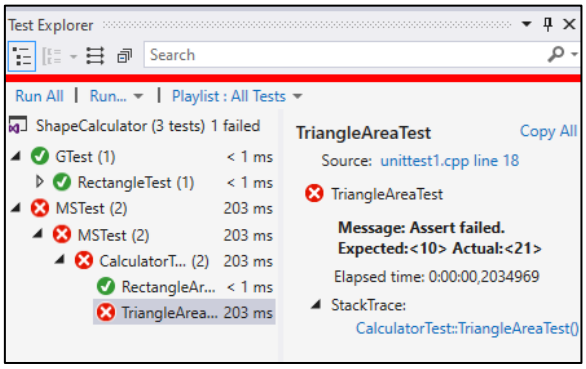
Die Tests selbst werden nach dem AAA-Verfahren erstellt. Das Verfahren teilt einen Modultest in drei verschiedene Teile. Im ersten Teil wird zunächst ein Ausgangszustand initialisiert („Arrange“). Hier werden also alle benötigten Objekte und Variablen deklariert. Im zweiten Teil wird dann die zu testende Komponente mit den zuvor festgelegten Objekten und Parametern aufgerufen („Act“). Im letzten Teil wird dann das Ergebnis der zuvor aufgerufenen Komponente mit dem erwarteten Resultat verglichen („Assert“).



Ein möglicher Test könnte nach diesem Prinzip beispielsweise so aussehen:

```
public class CalculatorTest
{
      [Fact]
       Public void CanAdd()
          {
                    //Arrange
                    var calculator = new Calculator();
                    int value1 = 1;
                    int value2 = 2;
                    //Act
                    var result = calculator.Add(value1, value2);
                    //Assert
                    Assert.Equal(3, result);
          }
};
```

Ausgewertet werden die Tests im integrierten Test-Explorer in Visual Studio. Der Test-Explorer zeigt in der folgenden Abbildung auf der linken Seite, welche Tests erfolgreich waren und welche nicht. Auf der rechten Seite wird beschrieben, weshalb ein Test fehlgeschlagen ist. Es ist möglich entweder Tests einzeln auszuführen oder einen kompletten Testdurchlauf aller Tests zu machen.



In welchem Umfang wird getestet?

Um sicherzustellen, dass jede Funktion ausreichend getestet wird, gibt es die sogenannte Testabdeckung (Code Coverage). Diese zeigt an zu wieviel Prozent der Programm Code von Tests abgedeckt ist. Ziel ist es, wenn möglich, eine Testabdeckung von mindestens 75% zu erreichen. Für das Generieren der Testabdeckung wird das Code-Coverage-Framework Coverlet verwendet. Um das Überprüfen der Testabdeckung zusätzlich in unsere CI/CD-Pipeline zu integrieren, wird das Tool Codecov eingesetzt.

2.2 Integrationstests

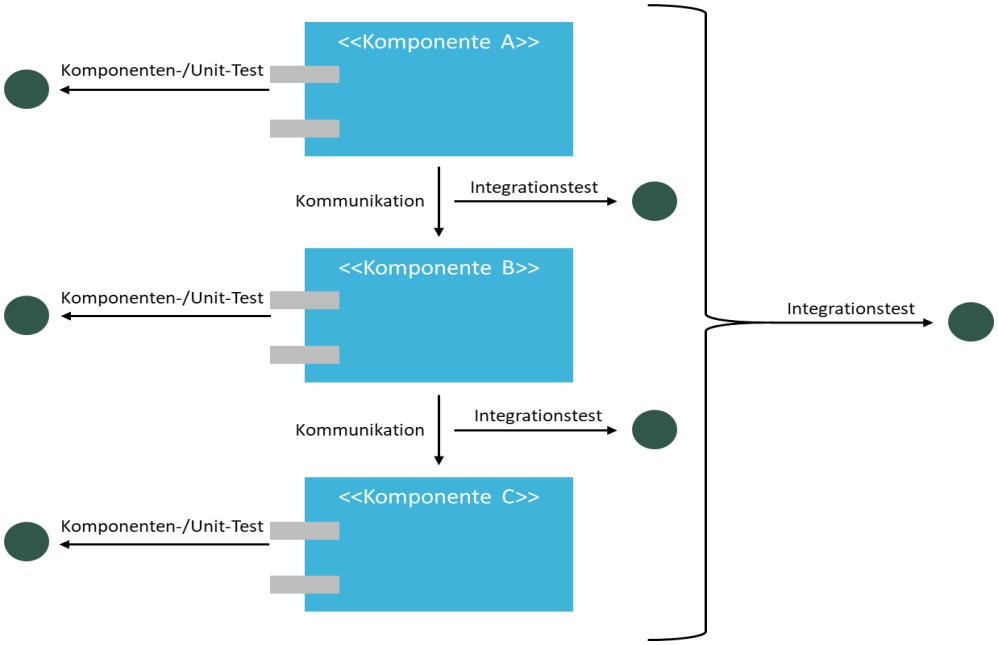
Integrationstests werden automatisch durchgeführt und erfordern mehr Aufwand als Modultests und sind länger in der Ausführung. Daher werden Integrationstests nur für die wichtigsten Szenarien, bei denen Komponententests nicht ausreichen, erstellt. Integrationstests werden idealerweise von einer Person die an der Komponente gearbeitet und einer Person, die diese noch nicht kennt, geschrieben. Auf diese Weise sollen sowohl White-Box-Tests als auch Black-Box-Tests erstellt werden.

Wie werden die Tests erstellt?

Für das Erstellen der Integrationstests wird, gleich wie bei den Komponententests, das Testframework xUnit verwendet. Da es auch hier nötig ist, verschiedene Abhängigkeiten zu lösen, wird ebenfalls das Mocking-Framework NSubstitute benötigt.

Wann wird getestet?

Die Integrationstests werden nach dem Bottom-Up-Prinzip durchgeführt. Das heißt sobald eine Komponente fertig gestellt ist und mit Hilfe von Komponententests getestet wurde, wird die Komponente gemeinsam mit den anderen bereits fertig gestellten Komponenten, verknüpft getestet. Zusätzlich werden alle Tests mindestens einmal am Ende jeder Woche ausgeführt. Die Ergebnisse dieser Tests werden standardmäßig dokumentiert und von einer weiteren Person gegengeprüft.



2.3 Systemtests

Bei den Systemtests wird unterschieden zwischen den Funktionstests und den Leistungstests. Funktionstests werden, wenn möglich, automatisiert und ansonsten manuell durchgeführt. Leistungstests sind nur optional und werden nur manuell durchgeführt.

Funktionstests

Die Funktionstests basieren auf den verschiedenen Produktfunktionen und Anwendungsfällen. Diese müssen einzeln auf ihre Funktionalität, aber auch auf das Verhalten im Fehlerfall getestet werden. Für jeden zu testenden Anwendungsfall wird ausführlich das Testziel, alle Voraussetzungen, die Eingabe, die erwartete Ausgabe und die verschiedenen Abhängigkeiten dokumentiert.

/T0120/	Interagieren	
/T0120/-A	Test Ziel Voraussetzung Eingabe Erwartete Ausgabe Abhängigkeiten	Überprüfung des Normalfalls Testperson befindet sich im Startraum des Test-Dungeons. Eingabe des Befehls „untersuche“ in das Textfeld der Spieloberfläche. Anschließendes drücken der „Enter“-Taste oder des „Senden“-Buttons. Beschreibung des Raumes -
/T0120/-B	Test Ziel Voraussetzung Eingabe Erwartete Ausgabe Abhängigkeiten	Überprüfung des Fehlerfalls Testperson befindet sich im Startraum des Test-Dungeons. Falsche Eingabe des Befehls. Anschließendes drücken der „Enter“-Taste oder des „Senden“-Buttons. Es erscheint die Fehlermeldung, dass der Befehl nicht existiert.

Nach diesem Schema wird für jeden Testfall genau festgelegt was passieren soll. Für alle Anwendungsfälle gelten folgende Voraussetzungen:

- Ein betriebsbereiter Computer mit einem modernen Web-Browser
- Eine aktive Internetverbindung
- Eine verifizierte E-Mail-Adresse

Wenn diese Voraussetzungen erfüllt sind, kann ein Anwendungsfall getestet werden. Während des Testens werden alle Informationen ausführlich protokolliert. Ein derartiges Protokoll ist wie folgt aufgebaut:

Test Case ID / Version	/T0120/	V1.0	
Testbedingung	Interagieren: Testperson untersucht mit dem Befehl "untersuche" einen Raum.		
Erstellt von/am	Daniel Kröker	14.04.2021	
Letzte Änderung von/am	-		
Beschreibung, Ziele für den TC	Überprüfung des Normalfalls		
Voraussetzung für Ausführung	Testperson befindet sich im Startraum des Test-Dungeons.		
Zustand nach Ausführung	Beschreibung des Raumes		
Weitere Anmerkung	-		
Benutzerrolle(n) für Testausführung	Dungeon Player		
Testausführung			
Getestete SW-Konfigurationsversion / MT-Gerätetyp			
Ergebnis	Erfolgreich	Mit Vorbehalt erfolgreich	Fehlgeschlagen
Normaler Ablauf			
Schritt Nummer	Benutzer Aktion	Erwartetes Ergebnis	Abweichung bei Testausführung
N1	Eingabe des Befehls "untersuche" in das Textfeld der Spieloberfläche und bestätigen mit der "Enter"-Taste oder des "Senden"-Buttons.	Beschreibung des Raumes	
Alternativer Ablauf			
Schritt Nummer			

Leistungstests (optional)

Mit Leistungstests wird überprüft, wie sich die Anwendung verhält, wenn viele Benutzer gleichzeitig Aktionen ausführen und viele Einträge in der Datenbank vorliegen. In unserem Fall wird mit den Leistungstests überprüft, wie sich die Anwendung verhält, wenn mehrere Benutzer gleichzeitig verschiedene Aktionen ausführen oder wenn mehrere Dungeons gleichzeitig aktiv sind. Richtige Stresstests bei denen tausende von Zugriffen gleichzeitig erfolgen sind nicht geplant, da der Aufwand solcher Tests die Ressourcen des Projektes übersteigt und in keinem Verhältnis zum Nutzen steht. Zudem ist ein solcher Fall von mehreren Tausend Spielern, die die Anwendung zeitgleich belasten, äußerst unrealistisch.

2.4 Akzeptanztests (optional)

Die Akzeptanztests bilden die Endstufe aller Tests und stehen daher unmittelbar vor der Inbetriebnahme. Bei diesen Tests geht es um die reine Funktionalität an der Oberfläche. Das Verhalten

sollte identisch zum späteren Live-Betrieb sein. Ziel ist es, mögliche Bedienfehler zu entdecken, um diese zu vermeiden, beispielsweise durch eine verständlichere Oberfläche. Durchgeführt wird ein Akzeptanztest von den Kunden oder je nach Wunsch auch von einer Gruppe an freiwilligen Testpersonen. Da diese Tests in unserem Fall keine wichtige Rolle spielen, sind sie nur optional und werden nur auf Wunsch des Kunden durchgeführt.

3. Technologien

- xUnit
- NSubstitute
- AutoFixture
- Coverlet
- Codecov

Versions-Historie

Version	Datum	Bearbeiter	Art der Änderung	Stand
1.0	08.04.2021	Leon Jerke	Erstes Aufsetzen des Dokuments	Fertiggestellt
2.0	15.04.2021	Team-Apollon	Besprechung des Entwurfs mit Herrn van Hoof	Fertiggestellt
3.0	17.04.2021	Leon Jerke	Verbesserung des Dokuments: - Automatisierung der Integrationstest in 2.2 - Änderung des Protokolls in 2.3 - Leistungstests und Akzeptanztests optional - Kleinere Verbesserungen an mehreren Stellen	Fertiggestellt
4.0	18.04.2021	Daniel Kröker	Qualitätskontrolle	Fertiggestellt