Testkonzept

Da bei der Entwicklung von Software Projekten mit zunehmenden Umfang das Finden und Beheben von Fehlern komplizierter wird, ist frühzeitiges ausfindig machen solcher durch rechtzeitige Tests angebracht. Dafür ist im Rahmen des Extreme Programmings ein Testzyklus vorgesehen, der aus 3 Teilen besteht:

1. Komponententests
   In dieser Testphase werden die einzelnen Methoden und Klassen auf ihre Funktionalität und auf Fehler geprüft. Dazu ist von den jeweiligen (soweit vorhandenen) Programmierpaaren passende Testreihen zu entwickeln und anschließend eine Dokumentation der Tests zu erstellen. Für die Tests ist die Nutzung von QUnit vorgesehen – eine einfach benutzbare Testsuite die für jQuery im speziellen aber auch für JavaScript an sich geeignet ist. Der Aspekt „Tests first“ des Extreme Programmings sieht vor, dass noch vor der Implementierung der Klassen und Methoden geeignete Tests zu schreiben sind. Diese Testklassen sollten demnach Test+Klassenname benannt werden.
   So wie die Klassen zu Paketen zusammen gesetzt werden, werden die entsprechenden Testklassen zu Testsuites zusammengeführt und Test+Paketname benannt.
2. Integrationstest
   In dieser Testphase wird das Zusammenspiel der einzeln entwickelten Klassen getestet. Somit ist es sinnvoll, vorher Zusammenhänge zwischen Klassen zu erkennen und das Zusammenwirken dieser dann zu testen. Dazu werden Klassen nach Geschäftsprozessfunktionalität integriert, getestet und erweitert.
3. Systemtest
   Nach erfolgreichem Integrationstest sind die Stories zu einer Zwischenversion des Produktes zusammenzuführen und einem Systemtest zu unterziehen. Dabei wird aus der Sicht des Anwenders geprüft, ob die Anforderungen an das Produkt erfolgreich umgesetzt wurden. Des weiteren sollten die einzelnen Gruppenmitglieder / Programmiergruppen die Zwischenversion eigenen Tests unterziehen, um so ein möglichst breites Spektrum an Tests durchzuführen.
   Bei Fehlern ist das Finden des Problems zu organisieren und die Behebung durch die Programmierer der beteiligten Klassen durchzuführen.

Dokumentation der Tests

Grundsätzlich sollte die Art der Tests festgehalten werden. Beim Auftreten von Fehlern sind diese, die verwendete Produktversion, die Quelle und ihre Lösung – wenn dann gefunden - festzuhalten. Auf diese Weise kann wiederholtes Auftreten des Fehlers einfacher behoben werden. Diese Dokumentation wird dabei von den jeweiligen Programmieren durchzuführen.

QUnit

Verwendung:
Um QUnit verwenden zu können, muss man jQuery, qunit.js und qunit.css einbinden und eine einfache HTML-Struktur zur Anzeige der Testergebnisse bereit stellen.

Einbinden von Qunit.css:
```
<link rel="stylesheet"
href="http://github.com/jquery/qunit/raw/master/qunit/qunit.css"
type="text/css" media="screen" />
```

Einbinden von qunit.js:
```
<script type="text/javascript"
src="http://github.com/jquery/qunit/raw/master/qunit/qunit.js"></script>
```

Um die Testergebnisse anzuzeigen, muss nun im Body-Tag der Seite folgende Zeile eingefügt werden:
<ol id="qunit-tests">

Die Ausgabe wird dann wiefolgt aussehen:
Name des Tests (Anzahl fehlgeschlagenener Tests, Anzahl erfolgreicher, Anzahl gesamt)

Ein Test sieht so aus:
test(name, *expected*, test)
Bsp:
```
test("Das ist ein Test", function () {
     expect(1);
     ok(true, "Dieser Test war erfolgreich");
});
```

Die entsprechende Ausgabe: Das ist ein Test (0,1,1)
Da in einem test()-Aufruf mehrere Tests durchgeführt werden können, wird nachfolgend auf der HTML Seite aufgeführt, welche Tests erfolgreich waren und welche nicht.

expected() ist optional und gibt an, wie viele Tests, sogenannte Assertions, zu erwarten sind.
Es gibt 3 Assertions:

- ok(Status, Nachricht) überprüft, ob das erste Argument wahr ist.
- equals(zu überprüfender Wert, erwarteter Wert, Nachricht) funktioniert wie ok(), gibt jedoch beide Werte aus und ist für nicht-boolean Werte gedacht.
- same(zu überprüfendes Objekt, erwartetes Objekt, Nachricht) funktioniert wie equals(), ist aber strikter, benutzt === und ist deswegen zum Vergleich von Objekten geeignet.

Des weiteren besteht noch die Möglichkeit, die Tests in Module zu unterteilen. Dazu muss lediglich
module("Name", [lifecycle]);
vor den Tests, die diesem Modul zugeteilt werden sollen, aufgerufen werden. Allen Tests von diesem Modul wird nun der Name des Moduls vorangestellt.
Es können in lifecycle setup- und teardown-Callback-Methoden initialisiert werden, die vor bzw. nach jedem Test in diesem Modul ausgeführt werden. So können zum Beispiel Testdaten für jeden Test erstellt werden.
Bsp:
```
module("Modul abc", {
     setup: function () {
          this.user = new User("TestUser");
     },
     teardown: function () {
          this.user = null;
     }
});
```