**Sprintzusazaufgabe Softwaretest**

**Aufgabe 1)**

Sie sollen ein Programm auf fachliche Richtigkeit prüfen, ohne dabei die innere Struktur des Programms zu betrachten.  
Welche der folgenden Testmethoden müssen Sie dazu verwenden?

**Antwort:** Black-Box-Test, da hier nur die funktionalität überprüft werden soll, ohne die tatsächliche Struktur oder verarbeitung zu betrachten bzw. zu bewerten.

**Aufgabe 2)**

Notieren Sie sich pro Testart eine markante Definition.

**Unit-Tests:**  
Werden von Programmieren vor dem Produktionscode als Spezifikation der untersten Ebene des Systems geschrieben. Wahrt die Intentionen des Programmierers bei der kontinuierlichen Integration. Idealerweise hundertprozentige Abdeckung, sollte aber im Bereich von mehr als 90% liegen.

**Komponententests:**Akzepttanztest der Business-Regeln. Erzeugt und prüft Inputdaten gegenüber der Outputdaten.  
Geschrieben von der Qualitätssicherung und Business, gelesen und interpretiert von Business.  
Deckt ca. das halbe System ab durch Standardfälle und sehr offensichtliche Grenz- und Ausnahmefälle sowie alternative Pfade. (Mehrheit der Grenzfalltests werden in Unit-Tests abgedeckt, sind hier bedeutungslos)

**Integrationstest**Sind nur bei größeren System sinnvoll, da diese aus vielen Komponenten bestehen. Es werden Komponentengruppen zusammengestellt und getestet wie gut diese miteinander kommunizieren. Die nicht zu testenden Komponenten werden durch Mocks und Testdoubles „simuliert“. Diese Tests sorgen dafür, dass die Architektonische Struktur fehlerfrei bleibt es können auf dieser Ebene auch Performance- und Durchsatztests anfallen. Integrationstest werden meist periodisch (nächtlich, wöchentlich, etc.) ausgeführt, da diese teils lange dauern.

**Systemtests**Automatisierte Tests, die gegen das gesamte integrierte System laufen.   
Systemtests testen Business-Regeln nicht direkt, sondern vielmehr ob das System korrekt verschaltet ist („Ultimative Integrationstest“), deswegen decken Systemtests nur etwa 10 % des Systems ab. Sie werden relativ unregelmäßig ausgeführt, je häufiger, desto besser. Bei diesen Tests werden auch Durchsatz- und Performancetests erwartet.

**Manuelle explorative Tests**  
Bei diesen Tests geht es darum herauszufinden, wie sich das System unter der Bedienung von Menschen verhält. Hierbei verzichtet man bewusst auf einen Testplan und Automatisierung. Manuelle explorative Tests prüfen auf unerwartetes und erwartetes Verhalten gleichzeitigt.   
Ziel ist es, möglichst kreativ so viele Macken wie möglich zu finden.

**Akzeptanztests**Tests die von Stakeholdern und Programmieren definiert wurden, um zu überprüfen, ob eine Anforderung erfüllt worden ist. Bevor mit der Implementierung eines Features begonnen wird, müssen die Akzeptanztests für dieses Feature bereit sein. Entwickler führen diese Tests aus, sehen wie diese scheitern und arbeiten anschließend daran das Feature zu implementieren, damit die Tests bestehen.

**Aufgabe 3)**

Zeigen Sie anhand eines konkreten Beispiels aus Ihrem Projekt auf, welche Testart Sie im Projekt verwenden.

**Antwort:** Wir könnten ein Komponententest erstellen. Dieser Test führt die Methode nach dem Login aus und überprüft, ob die eingeben Daten z.B. invalide sind und wenn ja, soll der Nutzer mit einem Hinweis darauf hingewiesen werden.

**Aufgabe 4)**

Ihre Aufgabe ist es zu beschreiben, wie Sie Fehlermanagement in Ihrem Programmier- bzw. Projektfluss einbauen.

**Antwort:** Sobald ein Fehler entdeckt wurde wird in Trello ein Task erstellt. Der Entdecker des Fehlers macht sich bereits Gedanken darüber, ob er diesen Fehler evtl. selbst beheben kann oder nicht. Spätestens im Daily-Meeting wird das Team auf den Bug hingewiesen und wenn nicht bereits geschehen weißt sich jemand diesen Bug zu, um ihn zu beheben. (Uns ist bewusst, dass dies nicht 100 prozentig konform mit Scrum ist, da Bugs nur im Notfall während dem Sprint zugewiesen werden sollen. Ansonsten landen sie ins Product Backlog.)

**Aufgabe 5)**

Beurteilen Sie, was mit folgender Grafik gemeint ist und was das Problem sein könnte.

**Antwort:** Dieser Entwickler meint, weil er keine Fehler gefunden hat gibt es keine weiteren Fehler mehr. Dies ist falsch, da nie sichergestellt werden kann, dass eine Software fehlerfrei ist.   
Außerdem neigt ein Entwickler dazu eine kleine „Blindheit“ zu entwickeln, wenn er Tests für seinen geschriebenen Programmcode erstellen muss. Da er die Methoden bereits kennt und weiß, was eingeben werden muss, um das gewünschte Ergebnis zu bekommen. Deswegen ist es immer gut, dass man den Code auch von einer anderen Person testen lässt.

**Aufgabe 6)**

Wie beurteilen Sie nachfolgende Aussage in Bezug auf die anschließende sichtbare Grafik?

**Antwort:** Wenn ein zusätzlicher Entwickler eingestellt wird, um die Softwarefehler effizienter zu entfernen hat der Tester mehr Aufwand, da er auch die Arbeit dieses Entwicklers testen muss. Und durch die Einstellung eines zusätzlichen Testers können weitere Fehler gefunden werden, wodurch die Software besser wird.

**Aufgabe 7)**

Stimmen Sie der Aussage von Edsger W. Dijkstra: „*Program testing can be used to show the presence of bugs, but never show their abscence!*“ grundsätzlich zu?

**Antwort:** Diese Aussage stimmt grundsätzlich schon, da das Testen der Software dafür da ist Bugs zu finden und es kann nie sichergestellt werden, dass eine Software komplett Fehlerfrei ist.

**Aufgabe 9)**

Beschreiben Sie die Bedeutung nachfolgenden Diagramms.

**Antwort:** Das Diagramm stellt die Kostenoptimale Qualität dar. Diese zeigt, dass ab einem gewissen Punkt es sich nicht mehr „lohnt“ viel Zeit in die Fehlervermeidung und Prüfung zu investieren, da diese im Laufe der Entwicklung immer Zeitaufwändiger wird und die Fehlerkosten die dadurch eingespart werden immer geringer. Deshalb sollte man nicht zu viel jedoch auch nicht zu wenig Zeit in die Fehlersuche bzw. Vermeidung investieren.