Software Process and Testing

Provide a brief definition of the term *software testing*.
Software Testing ist der Vorgang ein Programm mit der Absicht Fehler bzw. Bugs zu finden auszuführen.

Discuss the contribution of software testing in V-Model and Scrum. (hint: explain the contribution of testing by using small illustrations)
Das V-Modell hat schon sehr früh Testen als wichtigen Bestandteil in Form von eigenen Phasen in den Software-Entwicklungsprozess eingeführt. Dabei wird das System trotzdem noch in hierarchische Phasen gegliedert die nach einander durchlaufen werden. Als erstes wird die Spezifikation erstellt und getestet, als nächstes werden die Bestandteile der Komponenten getestet (Unit-Tests), danach die Komponenten und dann schließlich wird das Zusammenspiel der Komponenten und abschließend das gesamte System getestet. Neben den funktionalen Tests gegen die Spezifikation gehört auch ein Acceptance-Test dazu, der prüft ob die Spezifikation den tatsächlichen Bedürfnissen der Benutzer entsprechen.

Im Gegensatz dazu, ist Testing im Scrum-Modell ein inhärenter Bestandteil des Entwicklungsprozesses und kein ausgegliederter Prozess der danach statt findet. Durch das inkrementelle Vorgehen, welches Feature um Feature fertig stellt und ständig lauffähige Versionen produziert, finden die verschiedenen Tests die im V-Modell in verschiedenen Phasen stattfinden, häufiger und auch schon früher statt.
Scrum ist flexibler betreffend sich veränderter Anforderungen als das V-Modell.

Testing and Traceability

Explain the term traceability and provide a basic classification of traceability approaces.
Traceability bezeichnet die Eigenschaft die Geschichte über Entscheidungen und dazugehörige Informationen über bestimmte Bestandteile des Entwicklungsprozesses identifizieren und verifizieren zu können. (Warum wurde X gemacht? Wegen Y. → Falls Y nicht mehr gebraucht wird, gibt es eine bessere Möglichkeit X zu lösen? Etc.)

Traceability kann in drei Klassen eingeteilt werden:

- **Software-Architektur** auch **Vertikal**.
- **Implementierungsphase** auch **Horizontal**.
- **Versionsgeschichte** auch **Traceability over Time**.

Bei der Software-Architektur handelt es sich um die Beziehung zwischen einer Architektur-Ebene (System, Subsystem, Komponente, etc.) und eines Artefakttyps. (Welche Auswirkung hat Änderung X in der Komponente Y auf das System?)

Bei der Implementierungsphase handelt es sich darum, aus den verschiedenen Phasen die Kausalität herauslesen zu können. (Anforderung X → Implementierung von Y → Testfall Y_T)

Bei der Versionsgeschichte ist es wichtig herauslesen zu können, wann welches Feature hin zu kam um zu wissen welche Anforderung für welche Version gilt. So kann z.B. ermittelt werden welche Konfiguration für welches Feature ab welcher Version funktioniert und gebraucht wird.

Explain briefly what aspects are relevant regarding software testing and traceability. (hint: apply software testing to every basic traceability class)
Bei Testfällen die, die bestimmte Ebenen in der Software-Architektur betreffen, ist es wichtig nachvollziehen zu können wofür welches Feature gebraucht wird. Z.B. könnte für Feature X, welches eine Linuxkompatibilität benötigt, ein Schnittstelle zum X-Server geschrieben werden. Die Testfälle X_1, X_2 und X_3 testen ob diese Komponente funktioniert. Irgendwann wird entschieden, dass anstatt des X-Server eine andere Anbindung gewählt wird. Die Anforderung X wird weiterhin erfüllt auch wenn die Testfälle X_1, X_2 und X_3 fehlschlagen.
Nun ist es an der Zeit diese Testfälle zu entfernen, da herausgelesen werden kann, dass diese Komponenten nicht mehr gebraucht werden.

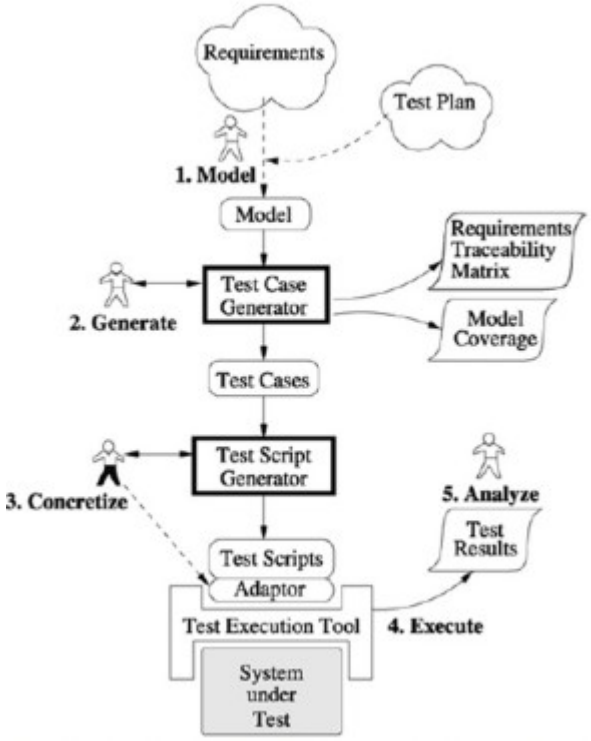
In der Implementierungsphase ist es wichtig, die Abhängigkeit zwischen den Testfällen und dem produzierten Code nachvollziehen zu können. So könnte z.B. nach einem Acceptance-Test eines Prototypen entschieden werden, ein bestimmtes Feature zu entfernen.
Durch eine gute Nachvollziehbarkeit ist es so möglich, alle betroffenen Code-Stellen zu entfernen wenn auf diese keine anderen Abhängigkeiten verweisen.

In der Versionsgeschichte ist ersichtlich, welche Testfälle für welche Version gelten. *? Keine Ahnung ob das das ist, was die hören wollen.*

Model-Driven Testing

What is a *model* and how does a model contribute to model-driven testing.
Ein Model ist eine Beschreibung und Abstraktion eines Systems. Es ist dazu da, um das System besser verstehen zu können und dessen (gewünschtes) Verhalten vorhersagen zu können. Da aus Modellen automatisch Testfälle generiert werden sollen, ist die Qualität des Modells entscheidend für die Qualität der so erzeugten Tests.

Provide the basic workflow of model-driven testing (e.g., by illustrating the basic concept graphically) and discuss advantages and disadvantages of model-driven testing.



1. Ein Modell des **S**ystem Under Test erstellen (SUT). Das Modell kann auch zusätzlich die Umwelt beschreiben bzw. nur aus der Umwelt des Systems bestehen.
2. Aus diesem Modell abstrakte Testfälle generieren lassen.
3. Die abstrakten Testfälle werden in ausführbare Skripte gegossen. → Konkretisiert
4. Konkrete Testfälle auf SUT ausführen und die Ergebnisse protokollieren.
5. Testergebnisse analysieren.

Vorteile	**Nachteile**

• Frühe Bug-Erkennung
  ◦ Modellieren deckt Probleme in der Spezifikation und im Design auf.
  ◦ Modellieren findet früh in der Entwicklung statt.
• Unterstützt sich entwickelnde Anforderungen und erleichtert die Testfallverwaltung.
  ◦ Ein Modell ist leichter zu aktualisieren als eine Menge von Testfällen.
  ◦ Ein Modell kann wieder verwendet werden auch wenn die Spezifikation sich ändert.
• Verringerte Kosten
  ◦ Testfälle werden aus Modell generiert → mehr Testfälle in weniger Zeit
• Es bleibt Zeit für fortgeschrittene Testprobleme.
• Verbesserte Arbeitszufriedenheit der Tester
	• Weiterhin hoher Aufwand. → vom Testen auf's Modellieren verschoben
	  ◦ Initialaufwand für Modellierung und Automatisierung
	  ◦ Exponential steigende Anzahl der möglichen Zustandskombinationen und wachsende Komplexität von großen Systemen.
	• Fortgeschrittene Fähigkeiten erforderlich bei Tester
	• Modell ist nur eine Abstraktion des Systems. Auch reflektierte Aussparung von Details kann dazu führen, dass wichtige Details nicht integriert werden.
	• Skaliert schlecht um von simplen zu komplexen Modell zu kommen. (um davon geeignete Testfälle für komplexe Sachverhalte ableiten zu können.)
	  ◦ Manuelles Testen kann bereits mit ein paar komplexen Testfällen komplexen Sachverhalt abbilden.
	• Hoher Aufwand um ein geeignetes Orakel (jemanden oder etwas, das bestätigt, dass das

	Modell korrekt ist) zu definieren.

Error → Fault → Failure

Explain the terms *error*, *fault* and *failure*, provide an illustrative example and discuss the *relationship* between the individual aspects.
Ein **Error** ist eine Aktion die von einem Menschen ausgeführt wird und zu einem falschen Ergebnis führt. (Kann auf der Seite des Administrators oder Programmierers auftreten.)
Ein **Fault** ist ein falscher Schritt, falscher Prozess oder eine falsche Datendefinition in einer Komponente oder dem System, welche dazu führen kann, dass eine Komponente oder das System davon abhält die verlangte Funktion auszuführen. Ein Fault der während des Betriebs auftritt kann zu einem Failure der Komponente oder des Systems führen.
Ein **Failure** ist eine Abweichung des Systems oder der Komponente vom erwarteten Verhalten bzw. Service oder Ergebnis.

Generell beschreiben die drei Fehlerstellen verschiedene Arten von Qualitäten. Ein Error bezieht sich auf die Prozessqualität. Ein Fehler der auftritt weil der Benutzer sich vertippt, die Zeit zu knapp wird, etc. Ein Fault bezieht sich auf die Produktqualität, also wenn Spezifikationen nicht überprüft werden, wenn das Testing nicht ordentlich durchgeführt wird, etc.
Ein Failure beeinflusst die Ausführungsqualität (Quality in Use). Also wie gut oder schlecht das Produkt vom Benutzer verwendet werden kann.

Coverage and Mutation Analysis
→ nicht ausgearbeitet, da der Graph in der Angabe fehlt.

Equivalence Partitioning and Boundary Value Analysis

```
/**
  * Returns true if a student is permitted to take part in
  * the exam. A student is only permitted if the following two
  * conditions hold true: (1) The student attended the course
  * and (2) the student achieved at least 35 points in exercises.
  *
  * @param points is the number of points achieved in exercises.
  * @oaram attended is true if the student attended the course.
  * @returns true if the student is permitted to the exam.
*/
boolean isPermitted( byte points, boolean attended);
```

Listing 1: Documentation and declaration of the function isPermitted

Apply equivalence partitioning and boundary value analysis for the input parameters of the function shown in the Listing 1. List all partitions you found including boundary values and mark invalid partitions.

	Points < 0 **Invalid**	0 <= points < 35 **Valid**	Points >= 35 **Valid**
true	false	false	true
false	false	false	false

Es steht nirgends ob 100 die Maximalzahl der Punkte einer LVA sind, also gehe ich hier davon aus, dass auch mehr möglich sind. Prinzipiell ist es natürlich möglich, dass es ab 100 Pkt. Wieder in einen Invalid-Intervall geht.

Derive tests cases from your equivalence partitions and boundary values for Listing 1, and report the test cases in the following table.

TC#	Points	Attented	Result
1	0	false	false
2	34	true	false
3	35	true	true
4	128	false	false

5	128	true	true
6	-1	true	false
7	-1	false	false

Implementing Test Cases with xUnit

```
public class FileTest {
	File f;

	@Before
	public void setUp() {
		f = new File("test.txt");
		assertTrue(f != null);
	}

	@Test
	public void testFile01() {
		assertTrue("Make sure the file exists", f.exists() );
	}

	@Test
	public void testFile01() {
		try {
			FileWriter w = new FileWriter("test.txt");
			for (int i = 0; i < 3; i++) {
				w.write("test " + i + ", ");
			}
			w.close();

			BufferedReader r =
				new BufferedReader(new FileReader("test.txt"));
			r.close();

			r.readline();		// throws IOException cause file is closed
		} catch (Exception e) {
			assertTrue(true);	// exception as expected
		}
	}
}
```

Listing 2: Example JUnit 4 Tests

Are the test cases in Listing 2 unit tests? Explain your answer.
Nein. Denn die Tests testen nicht das Verhalten von bestimmten Bestandteilen bzw. Units eines Systems sondern testen und erzeugen ein File.

What problems do you identify in the implementation of the test cases in Listing 2? Report and explain any problems you found.

- Das Fixture hat als undokumentierte Vorbedingung, dass ein File existiert. Dies sollte entweder per Testfall erzeugt werden oder dokumentiert werden.
- Das Fixture hinterlässt das System anders als es vorgefunden wurde. Mit jeder Testausführung wächst test.txt.
- Der Name des Files ist hardcodiert in jedem einzelnen Testfall.
- Die beiden Testfälle heißen gleich. → Compile-Time-Error (`könnte auch ein Tippfehler desjenigen sein, der die Angabe ins VOWI gestellt hat.`)

System Automation

Explain briefly the GUI testing approaches Capture & Replay and Script-based testing.

Capture & Replay bedeutet, dass der Testfall der so aufgezeichnet werden soll einmal manuell ausgeführt werden muss. Daneben läuft ein Programm, welches die einzelnen Schritte und das Ergebnis aufzeichnet.
Dies kann dann im Nachhinein verwendet werden, ob um zu prüfen ob das richtige Ergebnis zurück kam.

Script-Based Testing bedeutet, dass die Tester Skripte oder Programme schreiben müssen, die, die Aktionen ausführen und die Ergebnisse prüfen.

Capture&Replay hat den Vorteil, dass solche Testfälle einfach und ohne viel Wissen erstellt werden können aber den Nachteil, dass diese kaum anpassbar sind wenn sich am Programm etwas ändert. Script based Testing hat den Vorteil, dass sie leichter an sich ändernde Oberflächen adaptierbar sind. Allerdings ist der Aufwand höher und es bedarf Scripting-Kenntnisse.

Consider following situation: You are hired as a lead tester by a company that started developing a medium-sized ERP-Application in Java. Every month a new version of the application is released. For every release a large number of improvements and bugfixes are scheduled. In addition, product management insists, that with every release, at least one new feature is introduced. A new version needs to be fully tested before it can be released. So far, testing has been done manually by a team of three testers. They where running about 200 test cases via the GUI of the application.
Now you are expected to automate testing with a commercial test tool the company had already licensed. The tool supports Capture & Replay as well as Scripting of test cases in Java. On the one hand, project management proposes to use Capture & Replay, on the other hand, the development lead, argues for script based testing.

Prepare a list of distinct benefits for each of the two options. Explain each benefit briefly. Document any additional assumptions you make.

Capture & Replay	**Script-based Testing**
• Leicht zu erstellen, da die Tester nur die bereits bestehenden Testfälle manuell ausführen müssen. • Regressionstest ist inkludiert, da die Testfälle auf alten Versionen aufgenommen wurden. Geht der Testfall schief, ist etwas beim Update gebrochen.	• Testfälle können leichter angepasst werden als Capture & Replay Testfälle. • Regressionstest ist inkludiert, da die Testfälle für alten Versionen geskripptet wurden. Geht der Testfall schief, ist etwas beim Update gebrochen.