

Low Level

1. Finden Sie verständliche Erläuterungen der Begriffe *Test-Driven Development* (TDD) und *Behaviour-Driven Development* (BDD).
2. Erläutern Sie verschiedene Möglichkeiten wie Anwendungen getestet werden können: Auf welcher Ebene können Tests durchgeführt werden (Beispiel: GUI)?
3. Erläutern Sie *Test-Driven Development* (TDD). Was unterscheidet das TDD von nicht-testgetriebenen Ansätzen bei der Entwicklung von Funktionalität? Stellen Sie den Bezug zu der Aussage „Die Anwendung soll erst laufen, um die Fehlerbehandlung kümmern wir uns später“ her.
4. Erläutern Sie die Durchführung des Prozesses *Red, Green, Refactor*. Welche Aktivitäten werden in den einzelnen Schritten ausgeführt und wozu dienen sie?
5. Was sind *Spikes* im *Test-Driven Development*? Wann nutzt man Spiking und weshalb können Spikes auch ohne Testbett implementiert werden?
6. Recherchieren Sie Unit Testing-Frameworks für das .NET Framework. Durch welche Konzepte und Eigenheiten differenzieren sich diese Frameworks?
7. Was versteht man unter *Test Doubles* und wozu benötigt man sie? Welche Arten von Test Doubles gibt es? Welche Frameworks existieren mit denen man Test Doubles erstellen kann?
8. Erläutern Sie den Begriff *System Under Test* im Kontext von Test-Driven Development (TDD).

Mid-Level

9. Wie kann man den Begriff *Testbarkeit* definieren? Welche Voraussetzungen müssen Komponenten (z. B. Klassen) erfüllen, um Tests auf der Unit Test-Ebene zu ermöglichen? Hinweis: Was ist *Inversion of Control*?
10. Zeigen Sie an einem einfachen Codebeispiel einen Entwurf für Logik, die gut bzw. nur schlecht getestet werden kann. Zeigen Sie anhand entsprechender Unit Tests für jedes Beispiel, wo Probleme auftreten bzw. Testbarkeit nicht gewährleistet ist und wie man dies umgehen kann. (Hinweis: Sie können das angehängte Testprojekt nutzen und versuchen die Tests „grün“ zu implementieren. Die Tests müssen nicht unbedingt kompilieren bzw. erfolgreich durchlaufen.)
11. Was versteht man unter *Arrange, Act, Assert*? Welche Unterstützung für jede dieser Phasen existiert in verschiedenen Unit Testing Frameworks?
12. Wie kann der Prozess *Red, Green, Refactor* im *Pair Programming* (siehe: *Extreme Programming, XP*) durchgeführt werden?
13. Zeigen Sie an einem einfachen Codebeispiel eine gelungene und eine weniger gelungene Benennung von Testklassen und -methoden. Was sollte man bei der Benennung vermeiden? Hinweis: Die Benennung sollte u. a. Refactoring-sicher sein. Wie könnte die Benennung der Testklasse/-methoden für das Hinzufügen eines Artikels zum Warenkorb aussehen?

14. Erläutern Sie den Begriff *Code Coverage*. Wie kann Code Coverage gemessen werden? Welcher Wertebereich ist angemessen?

High Level

15. Was versteht man unter *Test-First*. Warum ist *Test-First* im *Test-Driven Development* (TDD) eine Designaktivität? Welche Vorteile ergeben sich daraus?
16. Was ist eine *Assertion* und wie werden Assertions von Unit Testing Frameworks unterstützt? Was spricht für oder gegen den Einsatz mehrerer Assertions in einer Testmethode?
17. Was unterscheidet *Behaviour-Driven Development* (BDD) vom *Test-Driven Development* (TDD)? Was sind in diesem Zusammenhang *Observations*? Welche Arten von BDD gibt es? (Hinweis: Man kann sowohl auf Unit-Ebene als auch Komplexe wie *User Stories* prüfen.) Welche Vor- oder Nachteile ergeben sich aus BDD gegenüber TDD?
18. Welches Vorgehen ist beim *Test-Driven Development* (TDD) empfehlenswert, wenn man einen Bug-Report für eine Anwendung erhält? Welche Vorteile bietet dieses Vorgehen, falls Regressionsfehler eingeführt werden? Hinweis: Erstellen Sie ein Diagramm das die Kosten für eines „klassischen“ und eines mit TDD entwickelten Projekts *auf lange Zeit* gegenüberstellt.
19. Was ist das *Object Mother* und das *Builder Pattern*? Wie können diese in Unit Test-Szenarien die Lesbarkeit des Codes verbessern? Zeigen Sie beide Pattern anhand eines Beispiels.