Ludwig-Maximilians-Universität München
Institut für Informatik
Prof. Dr. M. Hölzl, J. Rinke

Übungen zu Softwaretechnik: Programmierung und Software-Entwicklung: Lösungsvorschlag

Zusammenfassung der Socrative Fragen (Achtung: Diskussionsanstoß!)

Aufgabe Socrative-1 **1. Fragenblock - Lean & agile SW-Development** *Socrative*

a) Ziele (Vorteile) von Lean Software Development sind...

- ...klare Hierarchien in denen das Management das Entwicklerteam durch klare Vorgaben unterstützt.
- ...Beschleunigung der Prozesse und Reduzierung der Kosten indem das Management vorgibt was wann entwickelt wird.
- **...Beschleunigung der Kommunikation durch flache Hierarchien innerhalb des Teams.**
- **...Vermeidung fehlerhafter Entscheidungen indem sie nicht zu Beginn des Projekts, sonder zu späteren, passenden Zeitpunkten getroffen werden.**

b) Was ist ein primäres Ziel von Lean Software Development?

- **Entfernen aller Elemente, die keinen unmittelbaren Kundennutzen haben**
- Ausführlich und übersichtlich dokumentierte Software
- Hohe Anpassungsfähigkeit auf Veränderungen während des Entwicklungsprozesses
- Klare Rollenverteilungen innerhalb eines Entwicklerteams

c) Was ist ein primäres Ziel von Agile Software Development?

- Entfernen aller Elemente, die keinen unmittelbaren Kundennutzen haben
- Ausführlich und übersichtlich dokumentierte Software
- **Hohe Anpassungsfähigkeit auf Veränderungen während des Entwicklungsprozesses**
- Klare Rollenverteilungen innerhalb eines Entwicklerteams

d) Wovon hängt die erfolgreiche Umsetzung der Lean Principles stark ab?

- Von den Räumlichkeiten, die genug Platz für die individuelle Entfaltung der einzelnen Mitgleider des Teams bieten müssen.
- **Von der Zusammensetzung des Entwicklerteams und dessen Bereitschaft zur Umsetzung der Prinzipien.**
- Von den durch den Kunden formulierten Wünschen bezüglich der Umsetzung des Projekts.
- Von den gemachten Erfahrungen des Managements mit der Umsetzung der Lean Principles in industriellen Fertigungsprozessen.

e) User Stories haben häufig einen unterschiedlichen Umfang (Arbeitsaufwand). Welche Möglichkeit bietet sich dem Scrum Team großen Stories mehr Struktur zu geben?

- So lange mit dem Kunden, der die User Story formuliert hat reden, bis dieser die User Story wunschgemäßß verändert.
- Beim Management Beschwerde bezüglich zu umfangreicher User Stories einreichen.
- Zu umfangreiche User Stories möglichst niedrig priorisieren.
- **Umfangreiche Stories in mehrere kleine Tasks splitten.**

f) Ziele (Vorteile) von Lean Software Development sind...

- ...klare Hierarchien in denen das Management das Entwicklerteam durch klare Vorgaben unterstützt.
- ...Beschleunigung der Prozesse und Reduzierung der Kosten indem das Management vorgibt was wann entwickelt wird.
- **...Beschleunigung der Kommunikation durch flache Hierarchien innerhalb des Teams.**
- **...Vermeidung fehlerhafter Entscheidungen indem sie nicht zu Beginn des Projekts, sondern zu späteren, passenden Zeitpunkten getroffen werden.**

g) Für die Organisation des Product Backlog gilt, dass...

- **...der Product Owner mit dem Kunden Rücksprache halten und versuchen sollte die Items entsprechend dessen Wünschen zu priorisieren.**
- ...der Product Owner mit dem Entwicklerteam Rücksprache halten und versuchen sollte entsprechend dem geschätzten Aufwand zu priorisieren.
- **...der Product Owner versuchen sollte Abhängigkeiten bei der Priorisierung zu beachten.**
- ...der Product Owner möglichst viele Items in den Product Backlog eines Sprints nehmen sollte um den Kunden zufrieden zu stellen.

h) Während eines Sprints darf sich das Product Backlog nicht verändern, wahr oder falsch?

- true
- **false**

i) Einige User Stories sind sich ähnlich, was geschieht mit ihnen?

- Nichts, User Story ist User Story und mit ihnen wird das Product Backlog befüllt.
- Alle gefundenen User Stories werden thematisch sortiert und in das Product Backlog genommen.
- **Die User Stories werden nach Doppelgängern durchsucht und gefundene Doppelgänger werden entfernt.**
- Der Product Owner entscheidet welche ähnlichen User Stories er umgesetzt haben möchte und welche nicht.
- **Ähnliche User Stories können nochmal umfomuliert werden, da sie eventuell das Gleiche beschreiben.**

j) Wann sollten Akzeptanztests geschrieben werden?

- **Wenn PO und Entwickler sich über User Stories unterhalten und bestimmte Details festhalten wollen.**
- **Immer wenn neue Tests notwendig erscheinen.**
- Erst wenn während eines Sprints noch Kapazitäten frei sind.
- Nach Last Responsible Moment (LRM) soll erst nachdem eine User Story umgesetzt wurde ein zugehöriger Test geschrieben werden.

Aufgabe Socrative-2 **2.Fragenblock - Unified Modeling Language** *Socrative*

a) Was beschreiben Klassendiagramme und was beschreiben Objektdiagramme?

- Ein Klassendiagramm beschreibt die Kommunikation zwischen Klassen, wohingegen ein Objektdiagramm die Instanzen beschreibt.
- Ein Klassendiagramm beschreibt einen augenblicklichen Systemzustand, wohingegen ein Objektdiagramm die Kommunikation zwischen den Klassen beschreibt.
- Ein Klassendiagramm beschreibt die Systemarchitektur, wohingegen ein Objektdiagramm die Instanzen beschreibt.
- Ein Klassendiagramm beschreibt die Instanzen, wohingegen ein Objektdiagramm einen augenblicklichen Systemzustand beschreibt.
- **Ein Klassendiagramm beschreibt die Systemarchitektur, wohingegen ein Objektdiagramm einen augenblicklichen Systemzustand beschreibt.**

b) Welchem Modell würden sie Klassendiagramme und welchem Modell würden sie Objektdiagramme zuordnen?

- **Klassendiagramme: statisches Modell; Objektdiagramme: statisches Modell**
- Klassendiagramme: statisches Modell; Objektdiagramme: dynamisches Modell
- Klassendiagramme: dynamisches Modell; Objektdiagramme: statisches Modell
- Klassendiagramme: dynamisches Modell; Objektdiagramme: dynamisches Modell

c) Welcher Kategorie würden sie „Getter“ und „Setter“ jeweils zuordnen?

- Getter: Command; Setter: Command
- Getter: Command; Setter: Query
- **Getter: Query; Setter: Command**
- Getter: Query; Setter: Query

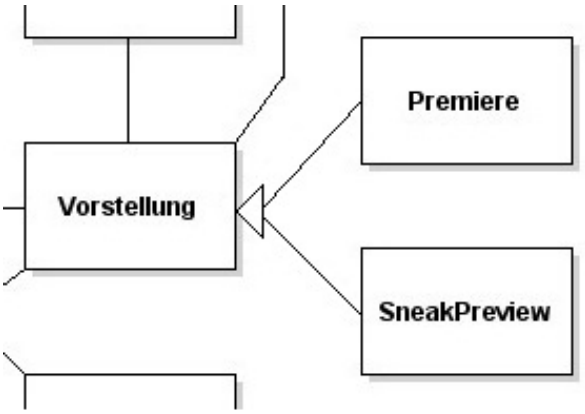

Abbildung 1: Generalisierung und Spezialisierung

d) Vorstellung ist eine Generalisierung von Premiere? (vgl. Abb. 1)

- **true**
- false

e) Premiere ist eine Generalisierung von Sneak Preview? (vgl. Abb. 1)

- true
- **false**

f) Sneak Preview ist eine Spezialisierung von Vorstellung? (vgl. Abb. 1)

- **true**
- false

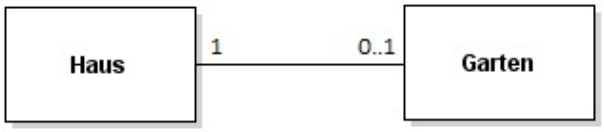

Abbildung 2: UML ⇒ *Java*

g) Es wird nur das Diagramm für die Klasse Haus implementiert. Vgl. Abb. 2

- **class Haus{private Garten garten;}**
- class Haus{private Garten[] garten = new Garten[1];}
- class Haus{private Set<Garten> garten;}
- class Haus{private Garten garten = new Garten();}

Abbildung 3: UML ⇒ *Java*

h) Es wird nur die Klasse Haus aus dem Diagramm implementiert. Vgl. Abb. 3

- class Haus{private Tuer[] tuer = new Tuer[1];}
- class Haus{private Tuer tuer = new Tuer();}
- class Haus{private Set<Tuer> tuer = new HashSet<Tuer>();}
- **class Haus{private Tuer tuer = new Tuer();}**

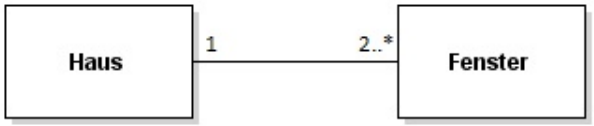

Abbildung 4: UML ⇒ *Java*

i) Es wird wieder nur die Klasse Haus aus dem gegebenen Diagramm implementiert. Vgl. Abb. 4

- class Haus{private Fenster[] fenster = new Fenster[2];}
- class Haus{private Set<Fenster> Fenster;}
- class Haus{private Fenster fenster1 = new Fenster();
  private Fenster fenster2 = new Fenster();}
- **class Haus{private Set<Fenster> fenster = new HashSet<Fenster>();}**

j) Das System wird am besten so umgebaut, dass die Klasse Vorstellung...

- ...als Interface mit den Methoden starten() und beenden() implementiert wird.
- **...als abstrakte Klasse mit den Attributen ührzeitünd "datumßusätzlich zu den abstrakten Methoden starten() und beenden() implementiert wird.**
- ...nicht verändert wird und alle anderen Vorstellungsarten von der Klasse Vorstellung erben.
- ...entfernt wird und durch die entsprechenden unterschiedlichen Vorstellungsarten ersetzt wird.

k) Vorteile der Verwendung von abstrakten Klassen sind:

- Man hat weniger Klassen in seinem System und bewahrt damit bessere Übersichtlichkeit.
- **Man vermeidet redundanten Code indem man die Methoden, die alle Unterklassen besitzen, in der abstrakten Klasse implementiert.**
- **Die Systemarchitektur erhält eine Struktur indem ähnliche Klassen als Unterklassen der abstrakten Klasse implementiert werden.**
- Man vermeidet die Verwendung von Interfaces und bewahrt sich dadurch einen übersichtlichen Programmierstil.

l) Hat ein Attribut einer Klasse die Sichtbarkeit protected so ist es...

- ...nur in allen Subklassen der Klasse sichtbar.
- ...überall sichtbar.
- ...nur im gleichen Package sichtbar.
- **...nur im gleichen Package und in allen Subklassen sichtbar.**
- ...nur innerhalb der Klasse sichtbar.

m) Interaktionsdiagramme modellieren...

- ...die Interaktionen zwischen den Methoden einer Klasse
- ...die Interaktionen zwischen den Klassen eines Packages
- **...die Interaktionen zwischen mehreren Objekten**
- ...das Verhalten eines Objekts

a) Bei MVC steht...

- **...das M für Model und repräsentiert die Daten und Änderungen auf den Daten**
- ...das M für Multiple und besagt, dass mehrere Objekte miteinander interagieren
- **...das C für Controller, der Interaktionen des Anwenders mit der View in Aktionen übersetzt**
- ...das C für Control und suggeriert, dass Anwender die Kontrolle über das Modell haben und die View ändern können

b) Vorteile durch Verwendung von Design Patterns sind:

- Reduzierung der Komplexität
- **Wiederverwendung von erprobten Lösungen**
- Erhöhte Performance
- Aufbrechen von Abhängigkeiten

c) Mögliche Nachteile durch Verwendung von Design Patterns sind:

- **evtl. erhöhte Komplexität**
- evtl. kennen Entwickler ein Pattern nicht, das führt zu Unstimmigkeiten
- evtl. fehlende Robustheit im Programm
- evtl. erhöhte Fehleranfälligkeit

d) Template Method ist ein...

- Creational Pattern
- Structural Pattern
- **Behavorial Pattern**

e) Ein weiteres Behavorial Pattern ist...

- Singleton Pattern
- Composite Pattern
- **Observer Pattern**

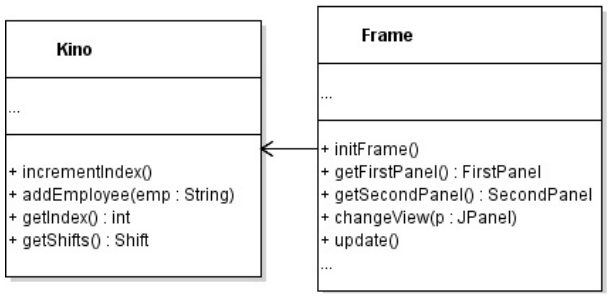

Abbildung 5: Observer-Pattern

f) Was muss konkret wie implementiert werden (Kino und Frame als Grundlage)? Vgl. Abb. 5

- **Kino muss Observable erweitern**
- Kino muss Observer implementieren
- Frame muss Observable erweitern
- **Frame muss Observable implementieren**

g) Wir betrachten die Strategie der Transparenz für die Implementierung der Methoden in Java Swing. Was passiert wenn eine Verwaltungsmethode, die für Composites gedacht ist, auf einem Leaf aufgerufen wird?

- Es gibt einen Fehler
- **Es passiert nichts**
- Der Aufruf wird an das Parent-Composite delegiert
- Das geht nicht, da solche Methoden nur in den Composites verfügbar sind

h) Intercessions sind Änderungen an der Objektstruktur der Meta-Ebene zur Laufzeit. Das heißt, dass eine Klasse X im laufenden System existiert, dass die Klasse X verändert wird und dass dann durch eine MOP-Funktion die veränderte Klasse X dem Metamodell hinzugefügt wird und die alte Klasse X ersetzt. Ist das in Java ohne weiteres möglich?

- true
- **false**

i) Mit Java Reflections ist es möglich einen Konstruktor, der die Sichtbarkeit private hat, außerhalb der Klasse aufzurufen!

- **true**
- false

j) Vorteile von Factories sind, dass...

- **...sie einen Single Point of Access für die Objekterzeugung bieten.**
- ...der Code leichter zu lesen ist, da die Erzeugung in der Factory stattfindet.
- ...es gibt keine, Factories sind ein Anti-Pattern.
- ...sie immer die Performance erhöhen, da die Instantiierung neuer Objekte in die Fabrik ausgelagert wird.

k) Sind Factories also immer eine Umsetzung des Factory-Method Pattern?

- **true**
- false

l) Ein Nachteil der Abstract Factory ist z.B.:

- Es gibt keine erkennbaren Nachteile
- Die Architektur ist sehr kompliziert und äußerst unübersichtlich
- **Neue Produkte lassen sich nur schwer hinzufügen**
- Es werden abstrakte Klassen verwendet

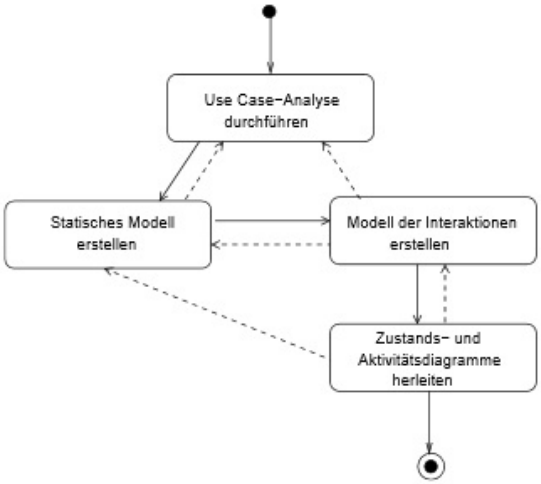

Abbildung 6: OOAD

a) Was für ein Diagrammtyp liegt hier vor? Vgl. Abb. 6

- Sequenzdiagramm
- Klassendiagramm
- Kommunikationsdiagramm
- **Aktivitätsdiagramm**
- Zustandsdiagramm

b) Am Ende der OOA steht...

- ...ein lauffähiger Prototyp.
- **...ein Systemmodell, das durch Diagramme beschrieben wird.**
- ...ein dynamisches Teilmodell, das Klassendiagramme enthält.
- **...ein dynamisches und ein statisches Teilmodell, auf deren Basis ein Systementwurf erstellt wird.**

c) Am Ende des OOD steht...

- ...ein lauffähiger Prototyp.
- **...die Softwarearchitektur als Vorlage für die Implementierung.**
- ...das fertige Produkt.
- ...ein dynamisches Teilmodell, das Sequezdiagramme enthält.

d) Was könnte das „Diamond-Problem“ sein?

- **Wenn das Verhalten einer Klasse, die auf unterschiedlichen Vererbungspfaden von ein und derselben Klasse erbt, nicht immer klar ist.**
- Eine Klasse, die von mehreren Klassen erbt, kann gleichnamige Attribute besitzen, wodurch nicht immer eindeutig ist, welches Attribut wann gemeint ist.
- Für eine Klasse die mehrere Oberklassen hat ist in der Klassebhierarchie unklar, welche Klasse die Vaterklasse ist.
- Die Kinder einer Oberklasse können nicht von einander erben.

e) Warum ist es manchmal sinnvoll der delegierenden Operation das aufrufende Objekt als Übergabeparameter zu übergeben?

- Dann ist immer klar wieso die Operation ausgeführt wird.
- **Dann hat das Objekt zu dem delegiert wird eventuell benötigte Informationen des Objekts, dass die Operation angestoßen hat.**
- Dann kann man leicht zurückverfolgen welches Objekt eine Operation angestoßen hat.
- Das macht überhaupt keinen Sinn.

f) Welches Pattern kommt zum Einsatz um in Java DB-Treiber zu laden?

- **Reflection**
- Template
- Observer
- Visitor

Aufgabe Socrative-5 **5.Verifikation und Testing** *Socrative*

a) Was versteht man unter Validierung und was unter Verifizierung?

- **Validierung: Entspricht Programm den Wünschen**
- Validierung: Entspricht Programm den Spezifikationen
- Verifizierung: Entspricht Programm den Wünschen
- **Verifizierung: Entspricht Programm den Spezifikationen**

b) Warum ist es sinnvoll innerhalb eines Projektes einen einheitlichen Workflow zu verwenden?

- Automatisierte Erzeugen von konsistentem Code
- **Vermeiden von Missverständnissen im Team**
- Eindeutige Fehlerzuweisung zu Entwickler
- Erhöhte Motivation der Entwickler

c) Was könnte in Zusammenhang mit Continuous Integration ein Nightly Build sein?

- Eine häufig verwendete Form von CI in der Entwickler vornehmlich nachts entwickeln
- **Eine Vorstufe von CI in der der Build Prozess nachts und automatisiert erfolgt**
- Nightly Build hat eigentlich nichts mit CI zu tun, das ist ein häufig auftretendes Missverständnis

d) Klassisches Fuzzing ist immer eine Form des...

- White-Box Testing
- **Black-Box Testing**
- Top-Down Testing
- Bottom-Up Testing

e) „Mocks“, „Stubs“, „Dummy-“ und „Fake-Objects“ werden in erster Linie bei...

- **Unit Tests verwendet**
- **Component Tests verwendet**
- Integration Tests verwendet
- Akzeptant Tests verwendet

f) Was ist der Unterschied zwischen Mocks und Stubs?

- **Mocks verifizieren simulierten Programmablauf, Stubs nicht**
- Stubs verifizieren simulierten Programmablauf, Mocks nicht
- **Ein Mock ist immer auch ein Stub**
- Ein Stub ist immer auch ein Mock