1	2	3	Sum
2	2	3,5	7,5

Houman Biglari

03.12.2015

Abgabegruppe Oktoberregen

Daniel Pujiula Buhl - Radu Coanda 344664 - Gregor Kobsik 359198 - Valentin Steiner 357980 - Jean Tekam

Aufgabe 6.1

a) Eine Person kann auf ein nicht existierendes Objekt zugreifen, was zu NullPointerExceptions führt, die wiederrum zu einem Programmabsturz führen können. Deswegen müssen diese abgefragt und abgefangen werden, was Zeit und Ressourcen kostet.

Des weiteren können so Fehler im System weiter propagiert werden, sodass eine Exception in einem anderem Programmteil auftritt, da der null-Wert der Methode weitergegeben wurde.
Dies führt zu Problemen beim debugging. √

b) Durch das Prinzip der Wiederverwendung von Programmteilen, werden die API's nicht nur von dem Entwickler, sondern auch von anderen Nutzern gebraucht. Diese kennen den inneren Aufbau der API's meistens nur spärlich und können somit unbedacht falsche Eingaben in das System einführen. Diese führen dann meistens zu Exceptions oder Bugs im Programm.

Eine defensive API überprüft jede Eingabe auf ihre Korrektheit innerhalb der Definitionswerte, die eine entsprechende Methode annimmt. √

c) Guava bietet die Möglichkeit von Optionals anstatt von null-Werten. Damit wird ein optionaler Wert anstatt keinem Wert weitergegeben, falls dieser Fall auftritt. Mit einem Standardwert kann das Programm weiterrechnen und stürzt nicht ab, oder es bekommt dadurch die Information, dass zuvor etwas schief gelaufen ist, dass repariert werden muss. √

Des weiteren bietet Guava die Möglichkeit von Immutables, also konstanten Ausdrücken, die nicht verändert werden können. Mit diesen ist sichergestellt, dass eine Methode während ihrer Ausführung sich sicher sein kann, dass der ihr übergegebene Wert nicht verändert wird und konstant bleibt. Dies kann vor allem bei parallel Computing vom Vorteil sein. √

d) fehlt! -2 Punkte

2 / 4 Punkte

Aufgabe 6.2

a) Logische Sicht: Die logische Sicht beschreibt die Funktionalität des Systems für den Nutzer. Für die Darstellung werden meistens Klassen-Sequenz-Diagramme verwendet.

Entwickler Sicht: In dieser Sicht ist das Augenmerk auf das Softwaremanagement gelegt, das Programm wird von der Sicht des Entwicklers betrachtet.
Hier werden meist Komponenten und Paketdiagramme verwendet.

Prozess Sicht: Diese Sicht beschäftigt sich mit dem Laufzeitverhalten des Programms.
Hier finden Aktivitäten Diagramme ihre Anwendung zur Darstellung.

Physische Sicht: In diesem Abschnitt wird die Systemarchitektur und die Verteilung der Softwarekomponenten ins Fokus gerückt. Diese ist vor allem für Systemarchitekten interessant.
In dieser Schicht finden Verteilungsdiagramme Anwendung.
Szenarien: in der letzten Sicht werden alle Sichten vereint und Anwendungsfälle, sowie mögliche Szenarien betrachtet. Hierbei wird auf die Zusammenarbeit in den Abläufen zwischen den Komponenten und den dazugehörigen Prozessen. Hierbei wird das Anwendungsfall Diagramm zur Darstellung benutzt.

√

b)

a. Benutzung von eindeutigen Variablenbezeichnern:

Die Faustregel ist, dass man den Code öfter liest als schreibt. Eine eindeutige und aussagekräftige Variablenbezeichnung unterstützt die Verständlichkeit maßgeblich und hilft den Programmfluss schnell und eindeutig zu verstehen. Den Variablen werden somit eindeutige Aufgaben bzw. Zustände oder Klassen durch die Namensgebung zugeordnet.

b. Kein Spaghetti-Code:

Man sollte keine „goto's" oder „return's" in der Mitte einer Methode benutzen. Diese führen zu einem unübersichtlichen Code, in dem oft und über weite Distanzen gesprungen werden muss. Hierdurch verliert man den Anschluss an die Verständlichkeit und kann den Programmfluss nicht verinnerlichen. Es führt auch oft zu einem unerreichbaren Code, wodurch die Funktionalität beeinträchtigt werden kann.

⇨ Insgesamt verhelfen Guidelines zu schnellem und gutem Informationsaustausch innerhalb von Teams. Einheitliche Styles machen das Lesen von Code einfacher und erleichtern das Einarbeiten in fremden Code. Dies sichert die Qualität der Entwicklung.

√

c)

a. „hohe Kohäsion“: Der Begriff definiert, dass jede Methode oder Klasse sei eindeutig für eine Aufgabe bestimmt, sodass eine klare Trennung innerhalb der Funktionalitäten festzustellen ist. Dies erfordert eine enge Zusammenarbeit der Methoden und Attribute. Dies erleichtert zukünftig die Wartbarkeit in dem Komponenten bestimmter eindeutiger Funktionalitäten ausgetauscht werden können, ohne andere Funktionen zu beeinträchtigen. √
b. „lose Kopplung“: Dieser Begriff definiert ein Konzept in dem Einzelne Elemente im System eine geringe Abhängigkeit voneinander aufweisen. Durch wenige Schnittstellen wird die Integration von solchen Komponenten in ein bestehendes System erleichtert. Zudem sind diese Komponenten leicht zu warten, da die Verbundenheit zu anderen Systemteilen sehr gering ist. √

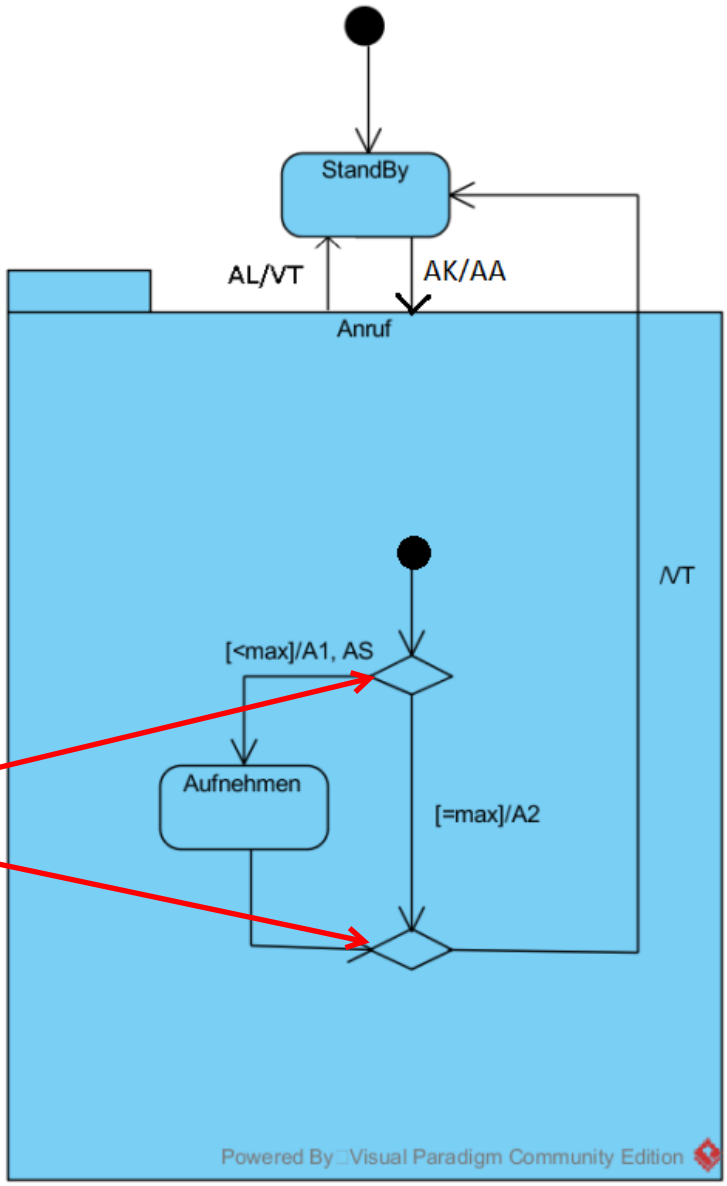
⇨ Insgesamt verhelfen beide Konzepte der Integrierbarkeit und der Wartung.

2 / 2 Punkte

Aufgabe 6.3

a)

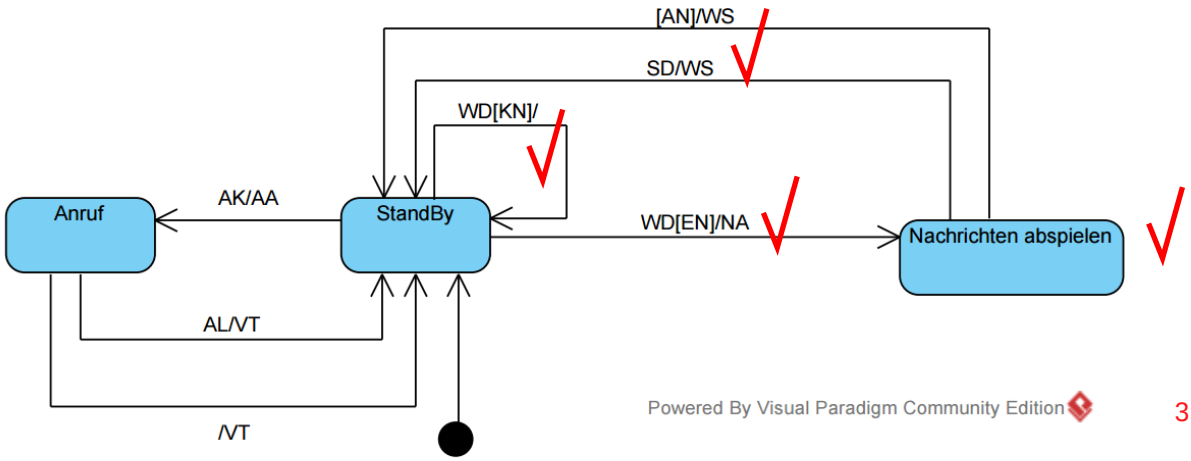
b)



Kein Bestandteil von Zustandsdiagrammen, es sollten sinnvollerweise stattdessen Zustände stehen! -0,5 Punkte

Aufgabe 6.3

c)



3,5 / 4 Punkte