Software Engineering Übung 8:

Gruppe:

1. Michael Scholz (matr.: 1576630, rbg: mi48azih)
2. Ulf Gebhardt (matr.: 1574373, rbg: hu56nifa)
3. Sebastian Weicker (matr.: 1625099, rbg: we87obyq)

Aufgabe 1: Software Design mit Hilfe von GRASP:

a)

Die nächste Flashcard während des Lernens auswählen:

Verantwortliche Klassen:
LernDialog → next()
→ showNextQuestion() (zählt index hoch, folglich wird hier die nächste Karte gewählt)
Controller: Contoller trifft nicht zu, da alle Controller-Funktionalität in der UI realisiert sind
Creator: Trifft nicht zu, da nichts erstellt werden muss.
Information-Expert: (FlashcardSeries,) Flashcard

Erstellen einer neuen Flashcard:

Verantwortliche Klassen:
FlashCardsWindow (Hinzufügen der Karte)
FlashCardEditor (Ausfüllen deer Karteninfos)
Controller: Contoller trifft nicht zu, da alle Controller-Funktionalität in der UI realisiert sind.
Creator: FlashCardsWindow → erstellt FlashCardEditor
Information-Expert: Flashcard

b)

Klasse	**Funktion**
Flashcard	Information-Expert (→ Falshcard enthält Flashcarddaten)
FlashcardSeries	Controller (→ verwaltet Falshcards) Creator (→ erstellt Flashcards)
Main	Creator (→ erstellt FalshcardsWindow)
Store	Creator (→ FlashCardSeries)
FlashcardsWindow	Creator (→ erstellt Falshcards)

Zum Vokabellernen sollen Frage und Antwort in allen Flashcards der aktuellen Serie vertauscht werden:

Verantwortlichkeit in LernDialog

Das automatische Generieren von Flashcards für das kleine Einmaleins, um nicht manuell 100 Flashcards erstellen zu müssen:

Verantwortlichkeit in FlashcardSeries

Aufgabe 2: Bewerten eines Software-Designs durch Betrachtung der Kopplung:

a)
Die Java Bibliothek stellt Standartfunktinalität zur Verfügung. Diese Funktionalität ist so allgemein, dass es nicht nötig ist sie in der Kopplungsanalyse zu beachten.
Außerdem steht diese Funktionalität in jeder Java-Version zur Verfügung, so dass keine Abhägigkeit besteht, die zu Portierungsproblemen führt.
Die Java-Klassen bedürfen keinerlei Änderungen, da sie bereits sehr gut und konsistent implementiert sind. Java-Lib-Funktionen werden nicht gelöscht (siehe deprected).

b)

Klasse	**Fan In**	**Fan Out**
Flashcard	5	0
FlashcardSeries	4	1
FlashcardsWindow	3	6

c)

- schwere Anpassbarkeit, da Änderungen in einer Klasse viele Änderungen in anderen Klassen nach sich ziehen
- geringe Verständlichkeit der Klasse, da der Kontext betrachtet werden muss
- schlechte Testbarkeit
- geringe Wiederverwendbarkeit

Bezug von Hoher Kopplung und Fan-In Fan-Out Metrik:

Die Summe von Fan-In und Fan-Out ergibt einen Wert, der eine Aussage über die Kopplung trifft. Je höher der Wert, desto höher die Kopplung.

Klasse mit hoher Kopplung aus der Tabelle von oben: FlashcardsWindow

Konsequenz häufiger Wiederverwendung:

Durch hohen Fan-In kann die Erweiterbarkeit der Software leiden, da alle Fan-In-Klassen ggf. geändert werden müssen.

Klasse mit hoher Wiederverwendung aus der Tabelle von oben: Flashcard

Aufgabe 3: Bewerten eines Software-Designs durch Betrachtung der Kohäsion:

a)

Funktion	**Felder**
getSeries()	series
getFrame()	frame
importFlashcardSeries()	FileDialog, series
SaveFlashcardSeries()	FileDialog, frame, series, file
saveAsFlashcardSeries()	FileDialog, frame, series, file
doSave()	series, frame, file
learn()	learnDialog
closeFlashcardEditor()	frame
CreateFlashcard()	FlashcardEditor, series
removeFlashcards()	Series, list
editFlashcard()	List, flashcardEditor, series

Zwei kohäsive Methoden: importFlashcardSeries() und SaveFlashcardSeries() mit den gemeinsam benutzen Feldern: FileDialog, series, file

Zwei nicht kohäsive Methoden: learn() und doSave().

b)

a=7
m=11

LCOM* = ((1/7(FIELDS)) - 11) / (1-11)
= ((1/7(24)) - 11) / (1-11)
= **0,757142857142857**

FIELDS= Sum(
frame: 5
list: 3
flashcardEditor: 2
learnDialog: 1
fileDialog: 3
series: 7
file: 3
)

Der berechnete Wert von 0,76 ist schlecht. Die Klasse hat mehrere Verantwortlichkeiten, die gegebenenfalls noch aufgeteilt werden können.