Abschlussbesprechung Software-Projekt II (Projekt-Realisierung) bei Prof. Knabe im WS 2016

Hier die in der Abschlussbesprechung am 08.02.2017 geäußerten Meinungen der Student(inn)en, gegliedert nach Themengebieten und Meilensteinen.
Legende: + positiv, – negativ bewertet, ! Wunsch, ~ teils/teils

Vorsemester Analysephase

!	Erster Durchstich wäre schon im Vorsemester sinnvoll gewesen, weil dann wesentliche technische Schwierigkeiten schon aus dem Weg geräumt werden könnten. Mehr Zeit, sich mit den Techniken auseinanderzusetzen, auch mit den im Plenum vorgestellten.

Meilensteine

4 Build-Werkzeug (Maven/SBT), Persistenzschicht, Testsuite, UI-Prototyp Obj.Verw.

+	Build-Werkzeug sinnvoll, Platzierung OK.
+	Parallele Entwicklung von UI und unterster Schicht sinnvoll wegen Zeitknappheit
+	Motivation durch Anschauung des UI-Prototyps
-	NoQueue: Parallele Testentwicklung erfordert viel Erfahrung, ansonsten zu häufiger Umbau.
~	PlanMi: Tests nicht parallel entwickelt, bei Bugs hat daher Testen über die Oberfläche sehr lange gedauert. Haben Wert der Tests erst am Ende gesehen.

6 1. Durchstich: Objektverwaltung 2er Fachklassen, Zuständigkeitsverteilung

!	PlanMi: Ticketsystem wurde nicht eingesetzt, wäre aber gut gewesen.
~	Zuständigkeitsverteilung bedingt sinnvoll, da es Verschiebungen geben kann.
+	Durchstich Objektverwaltung sinnvoll.
!	Erfahrung des Scheiterns ist nicht vermeidbar, aber lehrreich.

8 Zentrales Ausnahmemelden

+	PlanMi: Fand es gut, hatte sich früher nicht damit beschäftigt. Platzierung war OK.
?	NoQueue: Zentralisierung wichtig, Übertragung auf asynchrone Programmierung war Neuland.

10 2. Durchstich: Objektverbindungsverwaltung, JavaDoc-Spezifikationen

~	JavaDoc: Sollte bei größeren Teams Pflicht sein, hat aber nicht so gut geklappt, insbesondere bei Änderungen.
+	PlanMi: Passte gut in den Entwicklungsfluss. Beide Durchstiche gaben klares Ziel.
~	NoQueue: REST-API war unklar, da hätte ScalaDoc auch nicht geholfen.
+	NoQueue: Sinnvoll, obwohl sie den Termin nicht einhalten konnten.

12 Integrationstest

+	Selbst Durchklicken war gut.

13 Präsentation Gesamtsystem öffentlich

+	Schön, das Produkt der anderen Gruppen zu sehen.
~	PlanMi: Techniken der anderen Gruppe waren zu schwer zu folgen.

15 Abnahme

~	Mündliche Verteidigung aus Vorsemester erzeugte unnötige Befürchtungen vor der Abnahme.

Unterricht

	Sinnvoll?
	Stoffauswahl: Was interessant?

	NoQueue: Wie wird Bezug vom Stoff zum eigenen Projekt hergestellt? PlanMi: Folien mit Codeauszügen sind schwer nachvollziehbar, z.B. Control Abstraction, Generic DAO, besser wären komplette Mini-Projekte auf GitHub.
!	PlanMi: Technische Folien sollen ruhig ausführlich sein. Gesagte Erklärungen gehen schnell verloren.
-	NoQueue: Git ist mittlerweile überflüssig, da es im Studium schon dran kam. => Am Anfang des Semesters fragen, ob schon bekannt.
!	NoQueue: Projektplanung als Thema gewünscht: Ticketing, Aufgabenverteilung, dafür Beispielprojekt.
!	NoQueue: Kanban erklären.
!	PlanMi: Das Mock-testframework Mockito wäre sinnvoll zu behandeln. Im Plenum behandeln.
!	Automation von REST-API-Tests wäre ein gutes Thema.

Rücksprachen

+	Meilensteinrhythmus alle 2 Wochen? Gut, die wenigen dazwischen waren akzeptabel.
+	Anwesenheitspflicht bei Rücksprachen? War OK, auch weil nicht zu streng gehandhabt.

Gruppenarbeit

+	Selbstauswahl der Teammitglieder? OK
+	Gruppengröße 4-5? PlanMi: Konnten sich zu dritt gut treffen. Dreiergruppe hat funktioniert. NoQueue: Gab Probleme, sich an einem Ort zu treffen, daher nur virtuelle Treffen. Vierergruppe hat funktioniert.
-	Studentischer Projektleiter? Eher skeptisch dazu. Notendifferenzierung durch Projektleiter wäre zwar realitätsnah, aber dem Uni-Klima nicht angemessen.

Techniken

Versionierung und Projektkommunikationssoftware

Einsatz eines Persistenzframeworks

~	PlanMi: Hibernate: Anfangs seltsame Fehler wegen Dependencies. ID-Vergabestrategien Generated.IDENTIFIER hatte mit Derby nicht funktioniert, obwohl so dokumentiert, mussten Generated.AUTO benutzen.
~	NoQueue: Slick: Schwierig erweiterbar, z.B. Versuch, ein Generic DAO zu erstellen. Dokumentation war dünn, Userbase ebenfalls. Aber beeindruckende Funktionalität. War nicht möglich, eine Zelle in einer Zeile zu ändern, nur komplette Zeile. Waren teilweise genötigt, SQL zu verwenden. Subselects und komplexe Joins haben sehr gut funktioniert.

Einsatz eines UI-Frameworks

-	Vaadin: Programmierung in Java wirkte cool, Einarbeitungsaufwand falsch eingeschätzt. Dokumentation zu dünn, nur Codeschnipsel ohne Kontext. Würden es nicht wieder benutzen.
~	Ionic 2 +Angular 2: Zu knappe Fehlermeldungen, z.B. weißer Bildschirm. Gut: Große Community. Gute Angular-Doku. JS-Dependencies sind schwierig aufgrund von Kompatibilitätsproblemen. Layout nicht gut konfigurierbar. Etwas langsam. Vorteil: Gute Arbeitsteilung UI/backend möglich. Für einfache Sachen gut. Angular für größere Applikationen. React für kleine Geräte besser.

Einsatz von MulTEx

+	Vorgabe einer Struktur sinnvoll, Meldungstextgenerierung bequem. Konfiguration muss eingeführt werden (hatte Zeit gekostet). Wäre interessant, was sonst noch in großen Firmen verwendet wird.

Einsatz von Junit/ScalaTest-Testtreibersuite

Einsatz von

	Build-Werkzeug:

Studiengang

!	Technische Folien sollten als Nachschlagewerk verwendbar sein! D.h. die Erklärungen enthalten.
!	Mehr auf Tests eingehen.