5. Prozess: finale Reflektion

Freyschmidt, Henry Lewis (HLF)
Hama, Zana Salih (ZSH)
Krasnovska, Paula (PK)
Krüger, Lucas (LK)
Prüger, Marvin Oliver (MOP)
Seep, Tom-Malte (TMS)
Zabel, Steven (SZ)
Henry J. v. Rooyen (HvR)

18. Juli 2024

	Verantwortung	Inhalt	Grafik	Korrektur
Herausforderungen	SZ	SZ	SZ	HvR
angewandte Praktiken	SZ	SZ	-	HvR

Inhaltsverzeichnis

1 Herausforderungen

1.1 Prozess

1.1.1 Zeiteinteilung

Die Arbeit am Projekt war in Sprints unterteilt. Jeder Sprint bestand aus 4 Wochen. Es war also eine Herausforderung, die Zeit so einzuteilen, dass innerhalb der 4 Wochen die Planung des Sprints, die Implementation der geplanten Objekte und die Dokumentation des Fortschritts stattfinden kann. Folgendes ist am Ende der 4 Sprints unser allgemeines Ergebnis:

	Woche 1	Woche 2	Woche 3	Woche 4
Backlogs und Planung				
Änderungen am Backlog				
Planung des Sprints				
Feinentwurf				
Erstellung des Feinentwurfs				
Überarbeitung des Feinentwurfs				
Implementation und Tests				
Entwicklung von Tests				
Implementation				
Durchführung von Tests				
Dokumentation				
Installations-/Kompilieranweisung				
Dokumentation der Implementation				
Dokumentation der Tests				
Prozess-Dokument				

1.1.2 Aufwandsschätzungen

Zu den Aufwandsschätzungen war eine zentrale Herausforderung, wer an der Schätzung beteiligt sein sollte. Im ersten Sprint wurde versucht, dass alle sich an der Aufwandsschätzung jeder Task durch Planning Poker beteiligen, jedoch hatte das dazu geführt, dass einige Aufwandsschätzungen nicht qualifiziert waren und, nach einer Erklärung der Sachlage, sich entweder auf einzelne Personen verlassen wurde oder keine gemeinsame Schätzung abgegeben werden konnte. Aus diesem Grund wurden in späteren Sprints die Schätzungen auf die Teams ausgelagert, die jeweils für die Tasks zuständig waren. So wurden die Backend-Tasks beispielsweise ausschließlich vom Backend-Team eingeschätzt.

1.1.3 Test-Driven-Development

Beim Test-Driven-Development kam es zu der Herausforderung, dass es einzuschätzen war, was überhaupt getestet werden sollte und was auch ohne Test implementiert werden kann. Zu dieser Herausforderung gab es per se keine Lösung, welche beispielsweise durch eine analytische Planung hätte gegeben sein können (also dadurch, dass anhand von festen Aspekten für die Nötigkeit eines Tests argumentiert werden kann). Stattdessen wurde regelmäßig angemerkt, dass es nicht zu viele Tests braucht. Dass zu viele Tests entstanden war entsprechend ein Effekt dieser Herausforderung.

1.1.4 Anpassungen während der Implementation

Einige Male kam es während der Implementation dazu, dass der Plan, wie etwas implementiert werden sollte, umgestellt werden musste. Hier war es eine Herausforderung, darauf zu reagieren, sodass sowohl Backlog, Feinentwurf und die Stimmen aller Teammitglieder abgepasst wurden. Meistens war der Beschluss zur Anpassung während der Weekly Meetings, teilweise, falls nötig, in schnellerer Absprache im Chat, möglich und anschließend konnten die Dokumente ebenfalls entsprechend angepasst werden.

1.2 Team

1.2.1 Verfügbarkeit

Um die Verfügbarkeit des Teams sicherzustellen, wurde ein Weekly Meeting eingeführt. Dieses Weekly Meeting hatte einen festen Termin (montags, 17-19 Uhr) und hatte der gemeinsamen Abstimmung und Planung gedient.

1.2.2 Koordination und Aufgabenverteilung

Um die Aufgabenverteilung und die Koordination des Teams zu vereinfachen, wurden 4 Teams geschaffen: Product Owner, Frontend, Backend (Logik) und Backend (Datenbank). Die Teams Frontend, Backend (Logik) und Backend (Datenbank) waren mit 2-3 Personen, welche sich zu den Aufgaben abstimmten, besetzt. Der Product Owner war in seiner Rolle allein und hat Anpassungen mit allen Teams während der Weekly Meetings diskutiert. Die Teameinteilungen wurden unternommen, damit nicht alle Teammitglieder jede Technologie lernen müssen, sondern sich auf ihren Komplex konzentrieren können. Zudem wurde die Koordination verbessert, da es möglich war, Aufgaben den Teams zuzuordnen und die konkrete Aufgabenverteilung, also genauer welche Person an welcher Aufgabe sitzen soll, den Teams intern überlassen werden konnte.

1.2.3 Kommunikation

Anfangs war die Kommunikation die größte Herausforderung, da am Anfang des Sprints jedem Team Aufgaben gegeben wurden und am Ende des Sprints alles fertig sein sollte. Dazwischen gab es die Weekly Meetings, allerdings reichte das nicht aus, um gemeinsamen Fortschritt zu machen, da potentiell alle Teams an unterschiedlichen Aufgaben arbeiteten, sodass am Ende versucht wurde, alles zusammenzufügen und davor viel gewartet wurde, bis sich ein anderes Team mit einer Aufgabe befassen würde. Um dem entgegenzugehen, wurden im 3. und 4. Sprint Weekly Iterations eingeführt. So wurde zur Implementation der Aufgaben (im 2. und 3. Weekly Meeting) eine Iteration bestimmt, also eine Menge von User-Storys, welche in dieser Woche bearbeitet werden sollten. Das hat die Kommunikation verbessert und den gemeinsamen Fortschritt ermöglicht, da während einer Woche alle an den selben User-Storys gearbeitet hatten, welche dann im anschließenden Weekly Meeting besprochen werden konnten und daraufhin eine neue Iteration geplant werden konnte.

1.2.4 verschiedene Lösungsansätze

Falls es verschiedene Lösungsansätze gab, wurden diese innerhalb der Teams besprochen und sich dort entweder geeinigt oder die Frage in das Weekly Meeting gegeben. Auf diese Weise konnte immer eine Einigung erzielt werden.

1.3 Technologie

1.3.1 Apache Tomcat

Ursprünglich sollte Apache Tomcat zum Hosten des Servers verwandt werden, allerdings kam es dabei zu mehreren Problemen, da einiges an Wissen fehlte und, aufgrund schlechter Dokumentation, nur sperrlich erreichbar war. Da es so schwierig war, das Hosten hinzubekommen, wurde auf Spring beziehungsweise Spring Boot gewechselt, was das Hosten deutlich vereinfachte.

1.3.2 HTTP-Requests

Am Anfang wurden Get-Requests verwendet, welche an das Frontend geschickt wurden, wodurch einige Probleme entstanden, die durch Post-Requests gelöst werden konnten. Zu diesem Zweck wurde auf JSON gewechselt, was es ermöglichte, die Requests noch besser zu gestalten. Dadurch konnten Abfragen vereinfacht werden.

1.3.3 Hibernate

Hibernate unterstützt immer nur eine JPA PersistenceBean, wodurch eine Lösung geschaffen werden musste, welche es ermöglicht, mehr Informationen zu laden und gleichzeitig beschreibbar zu halten.

2 angewandte Praktiken

2.1 Sprints

Die Arbeit am Projekt war in Sprints unterteilt, wodurch es eine Woche Planung, zwei Wochen Implementation und Testing und eine Woche Dokumentation je Sprint gab. Diese Einteilung hat geholfen, um einen strukturierten Arbeitsplan zu bekommen und auf den gemachten Fortschritt entsprechend und optimal zu reagieren (durch die Dokumentation und Planungsphase). Zu Beginn war das Arbeiten in Sprints noch ungewohnt und hat dafür gesorgt, dass häufig der Zeitplan nicht ganz eingehalten wurde, beziehungsweise die einzelnen Aspekte des Zeitplans (beispielsweise Implementationszeit zu Dokumentationszeit) miteinander um etwas mehr Zeit konkurrierten. Mittels der Arbeit in Sprints wurde die Wichtigkeit von der Planung, Abstimmung und Dokumentation erkannt, da darauf basierend die jeweils nächsten Wochen stattfanden.

2.2 Test-Driven-Development

Das Test-Driven-Development hat es ermöglicht, einen Rahmen zu schaffen, welcher durch die Implementation umgesetzt sein sollte. Dieser Rahmen war eine gute Koordination, um die Implementation voranzubringen und zu wissen, ob die Implementation abgeschlossen ist. Das Team hat die Vorteile von Tests allgemein, aber auch davon, Tests vor der Implementation zu schreiben erkannt. Tests sind wichtig, um die Korrektheit und den Rahmen der Implementation festzulegen. Besteht dieser Rahmen bereits vor der Implementation, können die Tests auch als Anleitung für die Implementation genutzt werden. Probleme gab es zunächst beim Testen von HTTP-Requests, da diese neues spezialisiertes Wissen benötigten.

2.3 Pair Programming

Das Pair Programming wurde bei uns auf der Daten-Ebene und bei den Logic-Tests angewandt. Der Nutzen war, dass die Daten-Ebene und der Umgang mit der Datenbank möglichst konsistent und optimiert stattfinden sollte. Im Fall der Logic-Tests wurde es für ein koordiniertes Refactoring genutzt. Insofern hat das Pair Programming die Qualität erhöht.

2.4 Coding Standards

Coding Standards wurden eingesetzt, um über die genutzten Implementationen und Technologien hinaus eine möglichst verständliche Umgebung zu schaffen, da beispielsweise die Logik-Ebene sowohl die Daten-Ebene als auch das Frontend verstehen muss, beziehungsweise dort ansetzen muss.

2.4.1 Kommentare

Es wurde sich auf eine feste Kommentar-Struktur geeinigt, welche zu jeder geschriebenen Funktion anweist, einen größeren Kommentar mit den Angaben: Entwickler, Funktionalität und umgesetzte Task vom Backlog, zu hinterlassen. Wenn auch nicht immer genutzt, haben sich Kommentare dennoch als sehr hilfreich erwiesen, um Features zu besprechen, da man sofort wusste, an wen man sich wenden sollte, und um die Dokumentation anzufertigen beziehungsweise den Feinentwurf anzupassen, da die Funktionalität und die umgesetzte Task dazugeschrieben waren.

2.4.2 Bezeichner

Wichtig für die Arbeit mit Funktionen, welche man nicht selbst geschrieben hat und um sich besser zu merken, was eine Funktion tut, ohne sie selbst ständig erneut zu suchen und zu lesen, ist, dass die Namen der Funktionen beziehungsweise auch die Bezeichner die beim Aufruf und in der Funktion gebraucht werden, eindeutig und beschreibend benannt sind.

2.5 Backlog

Der Backlog umfasst User-Storys und Tasks, die die User-Storys weiter unterteilen. Dieser Backlog war der zentrale Bestandteil der Planung zum Projekt. Er wurde vom Team genutzt, um weitere Aufgaben zu finden und mittels der Aufgaben mit anderen Teams zu kommunizieren. Dadurch konnte also die Wichtigkeit von einer gemeinsamen Planung zum Beitrag der Koordination und Kommunikation erkannt werden.

2.6 Aufwandsschätzungen

Durch das Erstellen der Aufwandsschätzungen konnte ein Bewusstsein für die Dauer der Bearbeitung einer Task vertieft werden. Entlang der Sprints wurde dieses Bewusstsein konkreter und Tasks konnten preziser eingeschätzt werden. Zudem, durch die bewussten Einschätzungen, konnte anhand dieser eine Planung erfolgen, welche Tasks priorisiert werden sollten.

2.7 Weekly Meetings

Die Weekly Meetings dienten der Kommunikation und Planung mit dem gesamten Team. Dort wurde der gemeinsame Fortschritt zusammengetragen, diskutiert und beschlossen, was als nächstes passieren sollte. Die Weekly Meetings lernten dem Team, verständlich über ihre Implementationen sprechen zu können und Probleme akkurat einzuschätzen.

2.8 Weekly Iterations

Während der Weekly Meetings wurden Weekly Iterations bestimmt, welche alle Aufgaben beinhalteten, die in der nächsten Woche zu tun waren. Zudem wurde mittels der Weekly Iterations das Thema für das Weekly Meetings festgesetzt, da diese Aufgaben zu vergleichen und bezüglich ihres Fortschritts vorzustellen, Thema des nächsten Weekly Meetings sein würde. Diese Weekly Iterations haben sehr mit der Kommunikation geholfen, da jeweils wöchentlich das Team an den selben Aufgaben arbeitete und im jeweils nächsten Weekly Meeting zu einem eingeschränkten Teil der Aufgaben gesprochen werden konnte, statt dass alle Aufgaben durcheinander bearbeitet wurden von den jeweiligen Teams.

2.9 Retrospektive

Die Retrospektive diente der Reflektion des letzten Sprints. In diesem Meeting wurde beraten, welche Praktiken abgesetzt und welche hinzugefügt werden sollten. Mittels der Retrospektive wurden viele Probleme im Prozess gefunden und für zukünftige Sprints beseitigt. Das Team hat also gelernt, den Prozess um die Entwicklung einzuschätzen, zu besprechen und für ihre Zwecke zu optimieren. Durch die eingebrachten Praktiken (Team-interne Aufwandsschätzungen, Weekly Iterations, Coding Standards, ...) wurden bespielsweise viele Probleme bezüglich der Kommunikation gelöst.

2.10 Dokumentation

Die Dokumentation half dem Team, den Fortschritt je Sprint einzusehen und die Planung des nächsten Sprints anzupassen. Des weiteren konnte durch die Dokumentation nach außen kommuniziert werden. Also die Ergebnisse des Sprints klargestellt werden. Mittels der Dokumentation wurde also gelernt,

den Fortschritt so darzustellen und zu beschreiben, dass man nicht im Projekt involviert sein muss, um den aktuellen Stand nachzuvollziehen.

2.11 flexible Rollen

Flexible Rollen wurden als Praktik eingeführt, um Personen, falls sie mit ihrer Arbeit fertig sind, nicht zu zwingen, sich vom Projekt fernzuhalten. Stattdessen können sie sich in anderen Teams, die noch Arbeit haben, einbringen und helfen. Maßgeblich wurde diese Praktik zwischen Front- und Backend genutzt. Gerade für die Implementation des Controllers, welche Front- und Backend-Anbindung hat, war diese Praktik wichtig, um eine sinnvolle Arbeit zu ermöglichen.