Gruppenreflexion Team 2.A

Top 3 Erfolge

Aufrechterhalten einer harmonischen Arbeitsatmosphäre

Vermutlich das Wichtigste für eine erfolgreiche Entwicklung eines Produkts ist, dass sich alle beteiligten Entwickler gut verstehen und miteinander arbeiten können. In unserer Gruppe hat ein außerordentlich gutes Arbeitsklima geherrscht.

Begünstigt wurde unter anderem, dass bereits von Beginn an die wöchentlichen Meetings in Präsenz in der HTW stattgefunden haben. Im Gegensatz zu Plattformen wie Discord hat diese Form der Zusammenarbeit dazu geführt, dass wir mehr miteinander ins Gespräch kamen, uns besser kennenlernten und alle aktiv mitarbeiteten, weil das Feedback auf Aufgaben unmittelbarer und persönlicher erfolgte.

Ebenfalls (und vielleicht auch durch die analogen Meetings) war jeder motiviert, an der Entwicklung unserer Quiz-App beteiligt sein, und hat sich mit eigenen Ideen eingebracht. Dadurch konnten wir uns schnell auf einen groben Rahmen einigen, wie die App funktionieren soll und auf welchen architektonischen Strukturen sie gebaut wird. Dies hat besonders im Bezug auf die Realisierung von den ersten User Stories und -Tasks anfangs zu einem schnellen Fortschritt in der Entwicklung geführt.

Die breite Motivation zeigt sich auch dadurch, dass zugewiesene Tasks meist sehr schnell innerhalb der Iteration abgearbeitet wurden. So konnte der PO schnell die entwickelten Features ansehen und besser für die Vorstellung vor den Stakeholder vorbereiten, sowie sich über Weiterentwicklungs-Ideen Gedanken machen. Die Motivation führte sogar dazu, dass sich Teammitglieder freiwillig gemeldet haben, um mehr Tasks bearbeiten zu können. Dies hat sowohl den Fortschritt der Produkt-Entwicklung stark vorangebracht, als auch, die Arbeit für den Product Owner leichter gemacht. So hatte dieser weniger Tasks zu verteilen (da weniger zur Verfügung standen), und zusätzlich konnte dieser ein gut ausgearbeitetes Produkt den Stakeholdern vorweisen.

Dem Scrum Master das gute Arbeitsumfeld auch sehr viel Arbeit abgenommen. So hat die stetige allgemeine Motivation dafür gesorgt, dass sich alle in die Projektarbeit einbringen und eigenverantwortlich die Produktentwicklung mitgestalten. So musste der Scrum Master auch keine Konflikte zwischen Teammitgliedern lösen und konnte sich darauf konzentrieren, den allgemeinen Arbeitsprozess kontinuierlich zu optimieren.

Um den freundschaftlichen Umgang weiter zu fördern, hat unser Scrum Master auch einen gemeinsamen Billiard-Abend in den Sommerferien organisiert, wo wir uns noch besser kennlernen konnten und für noch mehr Motivation für die Arbeit im und für das Team gesorgt haben.

Drastische Änderung des Arbeitsprozesses hin zu besserer Umsetzung des Scrum-Prinzips

Wir haben uns wöchentlich getroffen, um über unseren Fortschritt zu reden, ähnlich zum Ablauf eines Weekly Stand-ups. Am Ende jeder Iteration gab es ein Sprint Review und anschließend eine Retrospektive immer mit dem gesamten Team.

Jedoch haben unsere "Stand-Up's" jede Woche 60-90 Minuten in Anspruch genommen, was überhaupt nicht mit der in der Vorlesung empfohlenen Menge von 15 Minuten übereinstimmt Dies hatte mehrere Gründe, daher ist dies auch als Misserfolg (siehe Schlechte Organisation der Weekly Meetings) gelistet und ausführlich beschrieben, da uns dies in der Entwicklung sehr eingeschränkt hat.

Dieses große Problem haben wir, wie im Misserfolg beschrieben, durch den Projektmanagement-Fachaustausch als solches erkannt. Und so haben wir in der darauf folgenden Iteration unseren Ablauf verändert: Nachdem unser Team neuen Input von den Stakeholdern in Form von User Stories erhalten hat, welche von einer Person während des Treffens protokolliert wurden, wurde unser Backlog, inklusive der neuen User Stories, im Team refinet. Dabei wurde die Komplexität der Items im Product Backlog geschätzt, sowie deren Priorität (stark beeinflusst durch den PO) festgelegt und im Status festgelegt, ob dieses Item 'Ready' ist, in der kommenden Iteration bearbeitet zu werden (also unsere 'Definition of Ready' erfüllt), oder noch im 'Backlog' verbleiben muss. Mithilfe dieser festgelegten 'Definition of Ready' kann dann der PO im Github-Board deutlich besser sehen, welche Aufgaben für die kommende Iteration zu erledigen sind. Nach diesem Review- und Refinement-Meeting erstellt der PO zahlreiche Tasks, abgeleitet aus den Items mit einer erfüllten 'Definition of Ready', sowie aus anderweitigen Anforderungen(z.B. Anwender-Dokumentation erstellen). So enstehen abhängig vom Umfang der Tasks ca. 3-6 Tasks pro Person pro Iteration.

In einem zusätzlich festgelegten Meeting über Discord, ein Tag nach dem Stakeholder-Treffen, werden die Tasks dann an das Team zugewiesen, damit es möglichst schnell wieder mit arbeiten beginnen kann. Dieses Treffen sollte nicht länger als 15 Minuten dauern.

Das Weekly Stand-up verläuft nun deutlich kürzer: Jeder im Team reihum spricht kurz an, was er bisher gemacht hat, und woran er demnächst arbeitet. Falls ein Problem besteht wird dies sehr kurz besprochen, oder es wird im Team eine Ansprechperson für die Problemlösung gefunden, an welche sich die Person wenden kann. Das einzige, was nun noch neu vergeben wird, ist die Bearbeitung von Bugs, falls sie eines dringenden Fixes bedürfen.

So konnten wir die Dauer der Meetings auf 15-30 Minuten einkürzen, womit wir sehr zufrieden sind. Auch sonst hat die Veränderung keine Probleme verursacht, sondern ist auf sehr positives Feedback im Team gestoßen, auch, da klar definiert ist, was jeder bis zum Ende der Iteration zu tun hat.

Wir finden, wir können Stolz mit uns sein, dass wir nach so vielen Wochen gemeinsamem arbeiten eingesehen haben, dass unsere bisherige Organisation nicht richtig funktioniert, und den Mut gefunden haben, diese komplett umzukrempeln. Ebenfalls, dass jeder im Team diesen Schritt mitgegangen ist und diese neue Meeting-Struktur mit einem zusätzlichen Meeting pro Iteration unterstützt hat, ist sehr lobenswert. Am Ende bleibt nur das Problem, das wir mit dieser neuen

Ordnung erst in unserer letzten Iteration anfangen konnten. So hätten wir vermutlich noch deutlich mehr schaffen können, wenn wir eher damit angefangen hätten.

Schnelle und frühe Organisation eines funktionierenden Code-Test-Systems

Womit wir zum Glück sehr früh angefangen haben, war das Beschäftigen mit sorgfältigem Code-Testing. So hat ein Teammitglied sich bereits nach dem ersten Praktikum, in welchem Tests behandelt wurde, dafür gemeldet, das Software-Testing übernehmen zu wollen. Da wir wussten, dass Software-Testing ein elementarer Bestandteil des Software Engineering ist, waren alle dafür, damit möglichst früh anzufangen. So hatten wir eine neue Rolle im Team: die Test-Verantwortliche.

Während andere Developer in unserem Team damit beschäftigt sind, Features zu implementieren, sorgt die Test-Verantwortliche dafür, dass der entwickelte Code getestet werden kann und somit wie von den Anforderungen gewünscht performt. Dafür entwickelt sie Integrations-und Unittests für den entwickelten Quellcode der Developer. Wenn ein Test Fehler ausgibt, hält die Test-Verantwortliche Rücksprache mit den Entwicklern des Codes, ob der Fehler auf der Seite der Tests oder doch im Quellcode liegt. Falls ein Fehler im Quellcode entdeckt wurde, wird dafür ein Bug-Issue auf Github erstellt. Die Testverantwortliche sorgt hauptsächlich für die Implementation von Unit-Tests und Integrationstests im Backend. Systemtests werden von allen Teammitgliedern beiläufig beim Durchklicken der Software, zum Beispiel durch Ansehen und Schnelltesten neuer Features, durchgeführt.

Da unsere Test-Verantwortliche über den gesamten Backend-Code verteilt arbeitet und testet, ist es nötig, dass die Code-Struktur des entwickelten Quellcodes in sich schlüssig und leicht verständlich ist, damit sie verstehen kann, was getestet werden muss. Somit werden ebenfalls die Entwickler dazu gedrängt, ihren Code gut zu dokumentieren und sinnvoll zu strukturieren bzw. zu modularisieren. Dies könnte auch hilfreich für die spätere Einbindung von neuen Backend-Entwicklern werden, falls dafür Bedarf besteht.

Das Wichtigste ist jedoch, dass nun sichergestellt wird, dass ein immer größerer Teil des Quellcodes korrekt funktioniert, wodurch sich die Entwickler mehr auf neue Produkt-Features konzentrieren können. Ebenfalls lässt sich beim Entwickeln von neuem Code nun schnell testen, dass der alte Code noch funktioniert, und nicht durch die neuen Eingaben Fehler auswirft.

Derzeit liegt die Testabdeckung bereits bei über 75% im Backend, womit wie sehr zufrieden sein können.

Top 3 Misserfolge

Fehlende systematisierte Architektur-Dokumentation

Für unser Softwaresystem wurde ein deutlich größerer Teil des Codes im Backend als im Frontend entwickelt. Eine systematische Architektur- und Code-Dokumentation wäre daher frühzeitig notwendig gewesen, wurde jedoch erst sehr spät eingeführt.

So hat dort initial eine Person das Django-Framework errichtet und dann bereits einige neue

Funktionen programmiert. Diese Person hat zwar die wichtisten Funktionen kommentiert, jedoch gab es immer noch einen hohen Einarbeitungsaufwand bei den dann neu hinzukommenden Backend-Entwicklern. Dies lag daran, dass zum einen das Django-Framework an sich ein großer 'Quellcode-Batzen' ist, der erstmal verstanden werden muss. Zum anderen hatten wir keine einheitichen Dokumentationsstandards, wie zum Beispiel Docstrings an jeder Funktion, da wir uns der Existenz dieser damals noch nicht bewusst waren, und, weil wir zu Beginn zu viel Wert darauf gelegt haben, einen vorzeigbares Softwareprodukt mit den entsprechenden Features zu entwickeln, wofür mehr Zeit in neuen Code investiert werden musste.

Dies hat dazu geführt, dass Entwickler, die erst später Code im Backend entwickeln sollten, sich nicht so recht getraut haben, diesen zu entwickeln. So wussten sie nicht genau, wo der benötigte Quellcode zu stehen hat bzw. wie viel davon bereits als wiederverwertbare Funktion einfach aufrufbar ist. So mussten die Entwickler sehr viel Zeit damit verbrauchen, zuerst den bestehenden undokumentierten Code zu verstehen, und kontaktierten dafür auch oft die Person, die den bestehenden Code entwickelt hat, um Ihnen beim Verstehen zu helfen. Oft haben die beiden Entwickler sich dann gemeinsam an den Code gesetzt, teilweise aber hatte dieser auch den Backend-Teil der Task vom unsicheren Entwickler komplett abgenommen, um den Zeitplan einhalten zu können. So ehrenwert es war, dass er die Aufgabe übernommen hatte, war dies jedoch langfristig nicht förderlich, da der unsichere neue Entwickler so nie gelernt hat, wie der Backend-Code funktioniert, und somit auch in Zukunft nicht in der Lage sein wird, diesen zu bearbeiten.

Diesem Problem hätte man vorbeugen können, wenn wir uns bereits früh auf Kommentier-Standards geeinigt hätten, was ebenfalls später in der Erstellung der Entwicklerdokumentation sehr hilfreich gewesen wäre. So wurde für diese Doxygen benutzt, wodurch die Einhaltung dieser Formatierungsstandards zwingend notwendig war, um eine übersichtliche Dokumentation zu erhalten.

Diese Kommentier-Standards zum Einen damals bereits ein bisschen mehr beim Verstehen des Code helfen können. Andererseits hätten wir auch bereits viel eher mit einer schnellen systematischen Erstellung einer Dokumentation des Codes beginnen können und sollen. Dafür spricht, dass die nun erstellte Entwicklerdoku laut unseren Entwicklern die Backend-Struktur sehr gut verdeutlicht und zu Beginn der Software-Entwicklung sehr hilfreich gewesen wäre, um den Code schnell zu verstehen. So hätten auch im späteren Prozess neue Backend-Entwickler keine große Hilfe von Entwicklern des bestehenden Codes benötigt, wenn wir eine solche Doku gehabt hätten. Dadurch kann dann neben der investierten Zeit der neuen Entwickler in das Verstehen des Code auch die investierte Zeit der bestehenden Entwickler in das Erklären des Codes verringert werden, auch wenn diese etwas mehr Zeit in die Dokumentation investieren müssen. Diese Zeitersparnis wird insbesondere langfristig mit dem neu hinzukommen jedes weiteren Backend-Entwicklers immer größer.

Dieses Problem mit der unzurechenden Dokumentation ist uns dann im Fachaustausch aufgefallen. So konnten wir erst spät (in der letzten Iteration) dieses Problem ausbessern, indem wir die Entwicklung von Features deutlich zurückschraubten und dafür die fehlende Dokumentation und Kommentierung des bestehenden Codes nachgeholt haben, nach klar festgelegten Richtlinien. Ebenfalls wurde eine Person festgelegt, welche sich mit einem Dokumentationstool wie Doxygen beschäftigen sollte, damit so der entwicklten Code verständlich, auch mit Diagrammen, erklärt werden kann.

So haben wir gelernt, dass eine systematisierte Dokumentation genauso wichtig ist wie

funktionierender Code.

Schlechte Organisation der Weekly Meetings

Ein Problem, was bereits früh aufgetreten ist und bis zur Veränderung unserer Meeting-Struktur durchgängig existent gewesen ist, war, dass die Stand-ups fast jedes Mal 60 - 90 Minuten in Anspruch genommen haben (wenn kein Sprint Review/Retrospektive oder ähnliches stattgefunden hat). Da wir uns gut verstanden haben, wurde dies in der Gruppe nicht als großes Problem angesehen, da die Zeit im regen Austausch schnell vergangen ist. Jedoch konnte man schon erkennen, dass etwas nicht ganz richtig lief, wenn das Meeting 4-6 Mal mehr Zeit in Anspruch genommen hat, als die ursprünglich in der Vorlesung empfohlene Menge von 15 Minuten.

So könnte doch bei Durchführen eines optimierten Stand-up allen beteiligten Teammitgliedern mindestens 60 Minuten pro Woche mehr Zeit für die Produktentwicklung zur Verfügung zustehen, wenn man die Stand-up-Dauer von 75-90 auf 15-30 Minuten verkürzt.

Der Grund für die Länge des Meetings lag daran, dass wir den Weekly Stand-up etwas falsch verstanden haben und nicht so durchgeführt haben, wie es eigentlich gemacht werden soll: So wurden für *jedes* Stand-up neue Aufgaben vom PO herausgesucht, die dieser aus gut passenden User Stories bzw. gefundenen Bugs hergeleitet hat. Falls Teammitglieder ihre Task zum Stand-up fertig gestellt haben, wurden ihnen von diesen erstellten Tasks neue zugewiesen, wobei noch einmal abgesprochen wurden, zu wem diese Aufgaben am Besten passen. Fall die Bearbeitung einer neuen User Story angefangen werden soll, wurde mit dem Team diskutiert, ob die vom PO herausgesuchte User Story vom Umfang her umsetzbar ist in der Task, bzw. welcher Developer dafür mitarbeiten müsste. Ein geordnetes Backlog Refinement der User Stories gab es bei uns nicht wirklich, die Priorität und Komplexität dieser wurde bei dieser Diskussion festgelegt. So wurde jede User Story einzeln in verschiedenen Stand-Ups refinet und die, welche es nicht in das Stand-Up geschafft haben, wurden gar nicht refinet, sodass auch eine vielleicht schnell umsetzbare User Story nicht bearbeitet wurde, wenn der PO diese nicht ausgewählt hat. Dies hat auch für eine unklare Struktur gesorgt, welche User Stories wie in der Iteration schaffen wollen, und wie viele wir zwischen den Sprint Reviews für die Stakeholder verwirklichen können

Da wir dennoch stetig Fortschritte in der Entwicklung von Features gemacht haben, schien uns die ineffiziente Bewertung von User Stories, sowie die lange Meeting-Zeit nicht als wichtiges Problem aufzufallen. Im Fachaustausch mit anderen Teams haben wir dann (endlich) herausgefunden, dass wir unsere Meeting-Organisation deutlich umstrukturieren müssen, wodurch wir dann um einiges effektiver Arbeit verteilen und erledigen können.

So hat unser Scrum Master den Ablauf unserer Meetings drastisch verändert, sodass unter anderem bereits am Anfang jeder Iteration im neu eingeführten Sprint Planning ein Backlog Refinement aller PBIs stattfand und im mächsten Meeting darauf basierend die Tasks verteilt werden, wobei der PO diese vorbereiten muss. Dies hat unser Stand-Up - Zeit deutlich verbessert auf nur noch ca. 20-30 Minuten pro Meeting. Diese Erkenntnis, das etwas falsch läuft, und die Einarbeitung in die neu organisierte Struktur empfanden wir aufgrund der guten Funktonsweise als großen Erfolg unseres Arbeitsprozesses. Daher ist diese in [Drastische Änderung des Arbeitsprozesses hin zu mehr scrum-artigen SE-Methoden] noch einmal detaillierter aufgeführt.

Hier besteht allerdings immernoch Verbesserungsbedarf, um auf die empfohlenen 15 Minuten zu

kommen. Daran müsste unser Scrum Master für die zukünftige Iteration noch arbeiten, wenn noch mehr Zeit für die Produktentwicklung wäre.

Stakeholder-Wünsche im Review zu wenig angefochten

Wir haben regelmäßig, am Ende jeder Iteration, ein Review mit den Stakeholdern durchgeführt. Dieses Treffen gab uns jedes Mal sehr konstruktiven und hilfreichen Input für die Entwicklung unseres Software-Produkts. Jedoch gab es oft, vor allem am Anfang, sehr viel Input. Dadurch haben wir nach dem Meeting meist darüber diskutiert, dass doch viel zu viel Entwicklungarbeit für manche Wunsch-Features zu investieren nötig ist, sie also praktisch in unserem kleinen Team nicht umsetzen ist. So haben wir viele User Stories gehabt, welche wohl nur sehr teilweise umgesetzt werden im gesamten überiterativen Entwicklungsprozess.

Zum einen hatten wir zu Beginn noch einen unausgereiften Product-Scope, durch welches wir noch nicht genau wussten, welche Features auch in unserem Plan stehen, noch zu implementieren, und welche Features überhaupt nicht zu unserer Software gehören sollten. Dies haben wir nach dem ersten Stakeholder-Treffen bereits erkannt, sodass wir dies ausbessern konnten und dann besser auf das nächste Stakeholder-Treffen vorbereitet waren.

Zum anderen jedoch wäre es Aufgabe des Product Owners gewesen, die Stakeholder-Wünsche einzugrenzen und auch mal 'Nein' zu den Feature-Wünschen zu sagen. Das wäre wichtig gewesen, um die zukünftigen Aufgaben des Teams schonmal vorab einzugrenzen, und es vor Überlastung zu schützen durch die übermäßige Menge an gewünschten Features, welche alle nicht wenig an Arbeitszeit benötigen. Jedoch war der PO dafür zu zögerlich, da die Stakeholder meist sehr überzeugt von ihren Wünschen waren und uns nicht klar war, wie weit wir ihre Wunsch-'Fantasie' beschränken können.

Nach Besprechungen mit anderen SE-Teams ist uns dann aufgefallen, dass wir auch die Stakeholder stoppen können und auch sollten, wenn sie zu viele Feature-Wünsche haben. Dies konnten wir dann in den letzten beiden Iterationen noch umsetzen, sodass der PO deutlich sagt, falls es genug Input ist, bzw. die Developer zu Wort kommen, falls sie erkennen, dass ein Feature zu viel Zeit in Anspruch nehmen wird. So konnten wir unser User-Story-Backlog in den letzten Iteration noch verkleinern, was auch für das Refinement wichtig geworden ist. So war es dort zum einen leichter zu entscheiden war, welche Priorität die User Storys haben, aber auch, wie sie zu schätzen sind, da dies mit weniger User Storys auch weniger Zeit in Anspruch nimmt.

Dennoch wäre es gut gewesen, die Stakeholder schon viel früher einzugrenzen, um weniger Zeit für die Diskussion über die Auswahl der User Stories zu investieren, damit das Team dann mehr Zeit hat, die wichtigen Features sorgfältig zu entwickeln

Einzelreflexion Carmina Langer

Mir war von Anfang an klar, ich in meinem Team mit Abstand die wenigste Expertise im technischen Bereich habe. Im Vergleich zu den anderen Mitgliedern des Developer-Teams hatte ich zu Beginn des Projekts deutlich weniger technisches Vorwissen. Besonders im ersten Semester fiel es mir schwer, aktiv an technischen Diskussionen teilzunehmen oder Verständnisfragen zu stellen. Obwohl wir noch längere wöchentliche Meetings hatten, in denen gemeinsam am Projekt gearbeitet wurde, habe ich mich zurückgehalten bei Fragen. Dennoch war es mir wichtig, einen erkennbaren und sinnvollen Beitrag zum Projekt zu leisten. Da das Dev-Team im Backend teilweise bereits vorgearbeitet hatte und ich dort nur schwer aufholen konnte, habe ich mich bewusst auf Aufgaben konzentriert, bei denen ich meine Stärken einbringen konnte. Im ersten Semester habe ich vor allem im Bereich Analyse und Entwurf gearbeitet. Dazu gehörten das Design und das konzeptionelle Grundgerüst der Website sowie die Erstellung von Diagrammen (C4 Modell, Domainmodell).

Erstes Semester

Ausgehend von den erarbeiteten User Stories und Personas habe ich mich intensiv in die Perspektive des Nutzers hineinversetzt. Da ich selbst regelmäßig mit Lernplattformen, Anki und Quiz-Apps arbeite, konnte ich eigene Nutzererfahrungen einbringen und konkrete Verbesserungsvorschläge entwickeln. So entstanden unter anderem kreative Konzepte wie das Maskottchen und die IQ-Points, die später Teil der Anwendung wurden. Ich habe mehrere Designvarianten für die Website entworfen, an denen sich die technische Umsetzung im weiteren Projektverlauf stark orientiert hat. Diese Arbeit lag klar im Bereich Software-Entwurf (Design) und bildete eine wichtige Grundlage für die nachgelagerte Implementierung durch andere Teammitglieder. Rückblickend war es für mich besonders motivierend zu sehen, dass unser Design bei den Stakeholdern sehr gut ankam. Parallel dazu habe ich im ersten Semester bereits zugesagt, das Testen im Projekt zu übernehmen (Qualitätssicherung). Aktiv umgesetzt habe ich diese Aufgabe jedoch erst im zweiten Semester, nachdem wir im Praktikum das Thema Testen behandelt hatten.

Zweites Semester

Im zweiten Semester habe ich mich dann verstärkt auf die Qualitätssicherung konzentriert. Dazu gehörte die Planung und Durchführung von Tests sowie die Dokumentation der Ergebnisse. Ich habe Testpläne erstellt, Testfälle definiert und die Tests selbst durchgeführt. Dabei habe ich verschiedene Testmethoden angewendet, angelehnt an die Testpyramide, die wir in der Vorlesung kennengelernt hatten. Zu Beginn der Testphase stellte sich heraus, dass mir meine geringe Erfahrung im Backend zunächst im Weg stand. Zusätzlich wurden im zweiten Semester noch größere Änderungen vorgenommen (z. B. der Wechsel von SQLite zu MySQL über einen Server),

weshalb ich mit einigen Tests bewusst gewartet habe, bis die Implementierung stabil war. Dadurch entstand zeitweise ein hoher Testaufwand in kurzer Zeit. Trotzdem konnte ich mich überraschend gut in den bestehenden Code einarbeiten. Durch die zuvor sehr ausführlichen Meetings wusste ich genau, welches Teammitglied für welche Teile zuständig war, und konnte gezielt Nachfragen bei Problemen. Dieses Vorgehen hat sich als sehr effektiv erwiesen. Nach den ersten gefundenen Fehlern entwickelte ich ein gutes Verständnis für das System und war schließlich in der Lage, Tests für Code zu schreiben, den ich selbst nicht implementiert hatte. Die gefunden Fehler habe ich in Github getrackt. [link: https://github.com/users/Gorg-tech/projects/7/views/4]

Bewertung und Lernerfahrungen

Meine Beiträge im Bereich Design, Konzeption und Testen haben aus meiner Sicht gut funktioniert. Besonders das frühe Design hatte einen nachhaltigen Einfluss auf das Projekt, da es die spätere Implementierung stark geprägt hat. Das strukturierte Testen hat mir gezeigt, wie wichtig Verständnis für das Gesamtsystem ist – auch ohne selbst alle Teile entwickelt zu haben. Rückblickend hätte ich mir gut vorstellen können, die Rolle des Product Owners zu übernehmen, da ich bereits viele Lernplattformen kenne und ein gutes Gespür für Nutzerbedürfnisse habe. Gleichzeitig bin ich froh, durch dieses Projekt neue Bereiche kennengelernt zu haben. Vor allem das systematische Testen haben mir sehr gefallen. Zukünftig würde ich mir eher zutrauen Aufgaben zu übernehmen, obwohl ich das technische Wissen nur teilweise habe. Ich habe gelernt, dass es oft wichtiger ist, aktiv zu sein und Fragen zu stellen, anstatt auf ein vollständiges Verständnis zu warten.

Einzelreflexion Eric Wolf

Ausgangssituation und Einordnung in den SE-Prozess

Im Projekt StudIQ habe ich als Full-Stack-Entwickler sowohl im Frontend, Backend als auch in der Infrastruktur gearbeitet. Unsere Arbeitsweise folgte einem **iterativen Ansatz** (Sprints), wobei wir **GitHub Projects** für das Backlog-Management nutzten. Meine Hauptaufgaben lagen in der Analyse & Entwurf (Datenmodell, Systemarchitektur), der Implementierung (Projekt-Setup, Scraper-Modul, Frontend-Teile) sowie technischen Entscheidungen und Deployment-Vorbereitung.

Ausgewählte Aufgaben

1. Technisches Projekt-Setup und API-Architektur

Zu Projektbeginn habe ich das initiale Setup für Django (Backend), Vue.js (Frontend) und die API-Verbindung übernommen (Issue #30). Dies ermöglichte dem Team, sofort mit echten API-Endpunkten zu arbeiten statt auf Mock-Daten angewiesen zu sein.

Der Lösungsweg orientierte sich an der **Schichtenarchitektur**: Django-Models und Serializer nach **Domain-driven Design** (Commit `1a9dca3`), REST-API mit ViewSets und Frontend-Service-Modulen (services/), sowie Umgebungsvariablen für sauberes Konfigurationsmanagement.

Ein wichtiger Meilenstein war die vollständige Definition der Client-Server-Schnittstelle (Commit `e3b8bb3`): Ich implementierte ein strukturiertes API-Service-Modul im Frontend mit zentraler Endpunkt-Verwaltung (`endpoints.js`), einem wiederverwendbaren API-Client mit CSRF-Handling (`client.js`) und Error-Management (`errors.js`). Parallel dazu passte ich das Backend-Datenmodell an die UML-Spezifikationen an und erweiterte die REST-API. Die Dokumentation im Server-README listete alle verfügbaren Endpunkte, HTTP-Methoden und Berechtigungen auf.

Learning: Ein minimaler "Walking Skeleton" ist extrem hilfreich für agiles Arbeiten. Die zentrale Definition und Dokumentation der API-Endpunkte in `endpoints.js` schafft Klarheit darüber, welche Schnittstellen verfügbar sind und wie diese zu nutzen sind.

2. Automatisierte Datenbeschaffung (Modulux-Scraper)

Um die App mit echten Inhalten zu füllen, entwickelte ich ein Scraper-Modul (modulux_scraper.py), das Studiengänge und Module der HTW Dresden aus der Modulux-Website extrahiert. Da keine offizielle API existierte und die Seite serverseitig rendert, musste ich das HTML parsen. Ein Migrations-Skript (populate_db.py) importiert die Daten strukturiert in die Datenbank und stellt Beziehungen zwischen Studiengängen und Modulen her.

Learning: Ich unterschätzte den Aufwand erheblich. Die HTML-Struktur-Analyse und

Datenbereinigung waren arbeitsintensiver als erwartet, für zukünftige Web-Scraping-Projekte muss ich realistische Zeitpuffer einplanen.

3. Frontend-Entwicklung und Responsive Design

Ich implementierte wesentliche UI-Komponenten wie die Navigation und die Startseite (Commit `c5d55a7`). Grundlage hierfür waren die **User Stories** und **Personas** aus der Anforderungsphase, sowie die Wireframes und Design-Konzepte aus dem Team. Ich setzte diese in Vue.js um, mit Fokus auf Responsivität durch Flexbox/Grid sowie die Umsetzung von Gamification-Elementen wie dem Pinguin-Maskottchen (Penguin.vue) und den Flammen (Streak-Indikator), um die Nutzermotivation zu erhöhen.

Challenge: Das Team arbeitete ohne einheitliches Design-System. Verschiedene Entwickler implementierten Features mit ganz unterschiedlichen Stilen, von minimalistische bis zu knalligen Gradients. Am Ende erforderte das ein großes UI-Refactoring, um alle Komponenten konsistent zu gestalten.

Learning: Ein konsistentes Design-System sollte früh definiert werden. Ohne gemeinsame Regeln (Farben, Abstände, Komponenten) führt zu viele parallele Entwicklung schnell zu Inkonsistenzen. Diese Standardisierung hätte uns am Ende viel nachträgliche Arbeit erspart.

Gesamtreflexion und Fazit

Erfolge:

- Der Umstieg von SQLite auf MySQL (PR #244) und das Deployment liefen reibungslos.
- Durch das schnelle Setup konnte das Team früh echte Features bauen.
- **Code Reviews** via Pull Requests etablierten sich als wichtiges Werkzeug zur Qualitätssicherung und Wissensverteilung.

Herausforderungen:

- Der häufige Kontextwechsel zwischen Backend und Frontend kostete viel Energie, fokussierte Blöcke hätten mir gutgetan.
- Wir hätten uns früher auf einen Dokumentationsstandard (z.B. Doxygen) einigen sollen. Da wir das erst spät definierten, mussten wir bestehende Funktionen und APIs nachträglich dokumentieren, ein zeitaufwändiger Prozess, der hätte vermieden werden können, wenn die Dokumentation von Anfang an parallel zur Implementierung entstanden wäre.
- Tests wurden erst spät in den Entwicklungsprozess integriert. Hätten wir von Anfang an **Test-Driven Development** (TDD) praktiziert und Tests in die GitHub Actions CI/CD-Pipeline eingebunden, wären viele Bugs früher aufgefallen.

Wichtigstes Learning: Ein gutes Fundament (Walking Skeleton, Tests, Dokumentierungen), klares Design-System und solide Test-Abdeckung zahlen sich am Ende massiv aus. Was anfangs wie "Umweg" wirkt, spart echte Zeit und Kopfschmerzen. Für zukünftige Projekte: Standards früh definieren, gemeinsam einhalten und so groß angelegte Refactorings von vornherein vermeiden :)

Reflexionsbericht Gabriel Wulkow Moreira da Silva: Frontend-Entwicklung und Prozessintegration im Projekt „StudIQ“

Im Rahmen des Moduls Software Engineering verstärkte ich das Projektteam ab der zweiten Projekthälfte als Frontend-Entwickler. Der Einstieg in ein bereits laufendes Projekt nach dem Scrum- Framework stellte eine besondere Herausforderung dar, da sowohl die technische Architektur als auch die organisatorischen Prozesse und die Teamdynamik bereits etabliert waren. Meine Aufgabe bestand darin, mich innerhalb kürzester Zeit in die bestehende Codebase einzuarbeiten, die Vision des Produkts zu verinnerlichen und produktive Beiträge zu leisten, um die finalen Meilensteine des MVP zu sichern.

1. Onboarding und Integration in bestehende Strukturen

Ein wesentlicher Teil meiner Einzelleistung bestand in der effizienten Einarbeitung. Um den Projekterfolg nicht durch lange Anlaufzeiten zu verzögern, wendete ich strukturierte Methoden des Reverse Engineering an. Ich analysierte die vorhandene Frontend-Architektur sowie die bereits implementierte Schichtenarchitektur (Django/Vue.js), um sicherzustellen, dass meine Beiträge technologisch konsistent blieben. Da die initiale Systemarchitektur zu Beginn nur teilweise dokumentiert war, nutzte ich die Weekly Meetings intensiv, um Wissenslücken zu schließen und mich mit den Backend-Schnittstellen vertraut zu machen. Dass ich bereits nach wenigen Tagen voll einsatzfähig war und erste User Stories im Sprint-Backlog übernehmen konnte, belegt mein tiefes Verständnis für modulare Softwarearchitekturen und agile Prozesse.

2. Relevanter Beitrag zum Projekterfolg

Trotz des späten Einstiegs habe ich zentrale, nutzerzentrierte Features verantwortet, die maßgeblich zur Fertigstellung des MVP beigetragen haben: Studiengang-Overview & Modulverlinkung: Dies war eines der komplexesten UI-Features. Ich implementierte die Logik, um Studiengangsdaten – welche über automatisierte Skripte im Backend bereitgestellt wurden – dynamisch darzustellen und die zugehörigen Module zu verlinken. Dies bildet das funktionale Herzstück der App. UI-Design & Konsistenz: Da in der ersten Projekthälfte noch kein einheitliches Design-System etabliert war, wiesen die Oberflächen stilistische Inkonsistenzen auf. Ich übernahm die Aufgabe, die UI zu harmonisieren. Ich stellte sicher, dass Farben, Abstände und Komponenten über alle von mir entwickelten Module (Settings, Freundesliste, Overview) hinweg einem

einheitlichen Standard folgten und trug so maßgeblich zur professionellen Außenwirkung der App bei. Settings & Personalisierung: Ich verantwortete den Einstellungsbereich, inklusive der Implementierung der Dark-Mode-Funktion sowie der sicheren Anzeige von Nutzerdaten. Anwenderdokumentation: Da ich als neues Teammitglied einen objektiven Blick auf die Software hatte, konnte ich eine besonders verständliche Anwenderdokumentation verfassen. Dies war ein entscheidender Faktor für die Qualitätssicherung, um die Usability-Anforderungen des Projekts zu erfüllen.

3. Fachlich korrekte Anwendung von Methoden und Techniken

Bei der Umsetzung meiner Aufgaben kamen fundierte Software-Engineering-Methoden zum Einsatz. Ich passte meine Arbeitsweise sofort an den bestehenden Scrum-Rhythmus an, schätzte meine Tasks realistisch ein und sorgte durch proaktive Kommunikation dafür, dass mein späterer Einstieg keine Blockaden für andere Teammitglieder verursachte. Technisch arbeitete ich konsequent nach dem Prinzip des Component-Based Development. Beispielsweise entwickelte ich die Link-Elemente der Modulübersicht so abstrakt, dass sie an anderen Stellen der App leicht wiederverwendet werden konnten. In der Versionsverwaltung achtete ich auf sauberes Branching und aussagekräftige Commit-Messages gemäß den Git-Flow-Vorgaben des Teams, um Merge-Konflikte im bestehenden Repository zu vermeiden.

4. Verständnis der Zusammenhänge im Software Engineering

Der Erfolg meiner Arbeit basierte auf dem Verständnis, wie Frontend, Backend und Dokumentation ineinandergreifen. Besonders deutlich wurde dies bei der Datenvisualisierung: Hier musste ich die Modellierung im Backend nachvollziehen, um die Daten im Frontend performant darzustellen. Ich kommunizierte eng mit der Backend-Entwicklung, um die Struktur der API-Antworten abzustimmen. Ein weiterer Fokus lag auf der Wartbarkeit. Da ich selbst fremden Code übernehmen musste, war mir bewusst, wie wichtig sauberer Code ist. Ich dokumentierte meine Funktionen direkt im Code und achtete auf eine klare Verzeichnisstruktur, damit zukünftige Entwickler meine Arbeit ebenso schnell verstehen können, wie ich die meiner Vorgänger.

5. Belegbarkeit der Beiträge

Meine Beiträge sind im Projekt durch die Repository-Historie (Commits zu Settings, Freundesliste und Overview) sowie durch das finale Nutzerhandbuch im Verzeichnis /deployement lückenlos dokumentiert und nachvollziehbar.

Einzelreflexion Georg Richter

Ausgangssituation und Einordnung in den SE-Prozess

Im Projekt habe ich die Rolle des Product Owners (PO) übernommen. Meine Hauptaufgabe bestand darin, die inhaltliche Steuerung der Produktentwicklung zu übernehmen und als Schnittstelle zwischen Entwicklungsteam und externen Beteiligten zu agieren. Dazu zählten insbesondere die Stakeholder, unser Coach sowie Prof. Anke.

Ich habe regelmäßig den Kontakt zu team-externen Personen gesucht und Sprint Reviews mit den Stakeholdern organisiert und moderiert. In diesen Reviews wurden die neu entwickelten Features präsentiert und gemeinsam evaluiert. Die Stakeholder erhielten dabei bewusst die Aufgabe, neue Funktionen selbst in der Anwendung zu finden und zu testen.

Auf Basis ihres Feedbacks wurde entschieden, ob eine User Story als abgeschlossen gilt oder weiterbearbeitet werden muss. Zusätzlich haben die Stakeholder noch weiteren Input für neue Features und somit neue User Stories geliefert. Diese wurden im Miro-Board festgehalten und dann von für unsere Entwicklung nicht sinnvollen Features bereinigt. Anschließend wurden diese von mir als User Stories formuliert.

Im Softwareentwicklungsprozess war ich vor allem den Bereichen Anforderungsmanagement zuzuordnen. Konkret umfasste dies neben den oben beschriebenen Sprint Reviews die Abstimmung mit Stakeholdern im Rahmen der Sprint Reviews, die Pflege und Priorisierung des Product-Backlogs, die Ableitung von Tasks aus User Stories und Organisation der Arbeit innerhalb einer Iteration sowie die Bewertung fertiggestellter Tasks im Teamkontext. Lange Zeit in unserem Entwicklungsprozess haben wir kein Refinement durchgeführt, wodurch nicht alle User Stories organisiert mit den Entwicklern geschätzt wurden. Daher habe ich die Schätzung ebenfalls anhand eigener Kriterien übernommen. Dass dies nicht der richtige Ansatz von Scrum ist, haben wir nach dem Projekt-Managements-Fachaustausch bemerkt, wodurch unser Scrum Master ein Backlog Refinement eingeführt hat, was mir die Schätzung abgenommen hat, weshalb ich mich dann neben der Priorisierung mehr mit der Pflege des Backlogs beschäftigt habe.

Ebenfalls habe ich, auf Grundlage des Product Backlog anfangs für jedes StandUp, mit dem Einführen des Refinements nur noch für jede Iteration konkrete Tasks für alle Teammitglieder erstellt und diese über GitHub Projects verwaltet. Die Tasks hat jedes Teammitglied benötigt, um zu wissen, woran es in der kommenden Zeit zu arbeiten hat. Hier ist auch eine klare Formulierung jeder Task sehr wichtig gewesen, damit ich mich als PO z.B. bei der Entwickluing eines Features darauf verlassen konnte, dass das Ergebnis meinen Erwartungen entspricht.

Fertiggestellte Tasks wurden im Weekly Stand-up vorgestellt. Gemeinsam mit dem Team habe ich das Ergebnis begutachtet und anschließend entschieden, ob die Task als erledigt gilt oder noch Nachbesserungen (z.B. Bugfixes) notwendig sind. Dabei wurden die funktionalen sowie die Qualitätsanforderungen an z.B. das entsprechende Feature überprüft, welche in der Task beschrieben wurden.

Kritische Bewertung meines Vorgehens

Trotz des stetigen großen Fortschritts unserer Software, habe ich an manchen Stellen Probleme gehabt meine Rolle als Product Owner korrekt auszuführen, was zu einigem an Mehrarbeit für mich und auch das Team gesorgt hat.

So finde ich, habe ich rückblickend habe ich meine Arbeit in Bezug auf das Stakeholder-Management nicht konsequent genug ausgefüllt. In den frühen Iterationen habe ich zu viele Feature-Wünsche der Stakeholder im Sprint Review ungefiltert aufgenommen. Dies führte zum Einen dazu, dass wir nach dem Sprint Review noch einmal bereden mussten, welche Features wirklich so umzusetzen sind und welche nicht im Scope unserer Software-Anforderungen liegt. Zum Anderen sorgte dies dafür, dass wir es gar nicht geschafft haben, immer alle Wünsche und Ideen im Miro-Board festzuhalten, wodurch sich dann an einige Features bei der User-Story-Erstellung nicht mehr erinnert werden konnte und diese ignoriert wurden. Ebenfalls befüllte die große Masse an neuen Features sehr schnell unser Backlog, was auch die Priorisierung erheblich erschwerte .

Ein Grund dafür war, dass die Produktvision und die Anforderungen zu Beginn noch nicht ausreichend konkretisiert waren. Dadurch fehlte mir eine klare argumentative Grundlage, um Wünsche abzulehnen, die nicht zum Produkt passten oder den Rahmen des Projekts gesprengt hätten. Ein weiterer Faktor war meine Unsicherheit, ob ein abgelehntes Feature möglicherweise von einzelnen Teammitgliedern gewünscht worden wäre. Aus Rücksicht auf das Team und die Stakeholder habe ich daher zu selten klare Entscheidungen getroffen.

Erst nach dem Fachaustausch zum Projektmanagement wurde mir bewusst, dass es explizit zur Rolle des PO gehört, klare Prioritäten zu setzen, Grenzen zu ziehen und diese Entscheidungen zu verantworten. In späteren Sprint Reviews habe ich – auch unterstützt durch das Team – deutlich klarer kommuniziert, welche User Stories umgesetzt werden und welche bewusst zurückgestellt oder verworfen werden. So kamen wir aus unserem letzten Sprint Review mit sehr viel weniger zu implementierenden Features aus, welche auch nicht noch einmal bereinigt werden mussten.

Dadurch habe ich gelernt, dass ich als PO mutiger auftreten muss, insbesondere im Gespräch über das Software-Produkt meines Teams, weil nur so eine fokussierte Produktentwicklung und eine realistische Umsetzungsplanung möglich sind.

Fazit

Zusammenfassend habe ich gelernt, meine Rolle als Product Owner aktiver und entscheidungsstärker auszufüllen. Insbesondere das konsequente Stakeholder-Management, sauberes Backlog-Refinement und eine strukturierte Iterationsplanung sind entscheidend für einen effizienten Entwicklungsprozess. Ich finde jedoch, dass ich zukünftig doch lieber in die Rolle eines Entwicklers schlüpfen würde, da ich die reine Code-Entwicklung deutlich angenehmer finde, als die ganze Planung für das gesamte Team und darüber hinaus übernehmen zu müssen.

Dennoch war es für mich eine wertvolle Erfahrung, auch mal die Sicht eines Product Owners kennenzulernen.

Reflexion Hans

Rolle im Projekt

Ich habe die Rolle des SCRUM-Masters übernommen. Zusätzlich gab es einen Product Owner, sodass die Rollen im SCRUM-Team grundsätzlich eindeutig verteilt waren. Zu Beginn hatte ich jedoch Schwierigkeiten, mit meiner Rolle und mit den damit verbundenen Aufgaben richtig umzugehen. Besonders in den ersten Iterationen fiel mir das schwer. Trotz meiner Rolle als SCRUM-Master war es mir wichtig, auch selbst zu programmieren. Daher habe ich mich schrittweise in das Developer-Team eingearbeitet. Ich hatte glücklicherweise sehr engagierte Menschen im Team, die für Fragen jederzeit offen waren

Beitrag als SCRUM-Master

Als SCRUM-Master lag mein Fokus auf der Organisation des Entwicklungsprozesses und von Anfang an auf einer positiven und offenen Arbeitsatmosphäre. Zu Beginn des Projekts habe ich den Entwicklern bewusst viele Freiheiten bei der technischen Umsetzung gelassen, da ich selbst weniger technisches Vorwissen hatte. Diese Vorgehensweise funktionierte zunächst gut, um schnell eine Grundlage zu schaffen, die später weiterentwickelt und angepasst werden konnte. Ein wesentlicher Beitrag meinerseits war es außerdem, mich dafür einzusetzen, dass Meetings regelmäßig in Präsenz stattfinden. Auch wenn die Meetings im ersten Semester teilweise länger dauerten als im klassischen SCRUM vorgesehen, waren sie für den Teamzusammenhalt sehr wertvoll. Im Verlauf des Projekts habe ich für dieses Vorgehen mehrfach positives Feedback erhalten. Zudem hatten wir in unserem Team zwischenmenschlich keine nennenswerten Probleme. Im weiteren Projektverlauf zeigte sich jedoch, dass das technische Verständnis nicht automatisch aus den Präsenzmeetings hervorgeht. Obwohl wir uns wöchentlich über eine Stunde getroffen haben, kam es zu Problemen, wie z.B. das Developer gleichzeitig eine Sache implementiert oder gefixed haben, während anderen das Verständnis fehlte zur grundlegenden Funktionsweise der Anwendung. Für mich war der Fachaustausch im zweiten Semester ein Wendepunkt. Durch den Fachaustausch musste ich mich kritisch mit unserer Arbeitsweise auseinandersetzen. Beim Austausch mit den anderen fielen vor Allem folgende Punkte auf: wir machen kein Refinement Meeting, unsere Meetings dauern zu lange, wir brauchen mehr Struktur und Austausch im Backend (teilweise ungenutze Variablen oder Dopplungen, SQLite Datenbank auf Github) und wir übernehmen zu viele User Stories auf einmal, die nicht fertig werden können in einer Iteration. Nach dem Fachaustausch habe ich mit dem Product Owner und einzelnen Teammitgliedern gesprochen und mir ihre Meinung zum aktuellen Stand des Projektes und der Arbeitsweise eingeholt. Basierend darauf haben ich mit dem PO unsere Meetingstruktur angepasst. Es gab nun Refinementmeetings und die Tasks für 2-3 User Stories wurden zu Beginn der Iteration festgehalten. Der nächste Punkt war das Aufräumen des Backends. Zusammen mit einem der Backendspezialisten bin ich mich mehrfach über Discord den Code durchgegangen. Besonders die API-Endpunkte habe ich mir ausführlich erklären lassen. Innerhalb der Meetings habe ich zusätzlich Verständnisfragen gestellt, welche nicht nur mir geholfen haben, sondern auch anderen Teammitgliedern mit weniger technischem Vorwissen. Zu diesem Zeitpunkt habe ich die Sequenzdiagramme unsrer wichtigsten Funktionen erstellt, was mir und anderen zusätzlich geholfen hat den Ablauf zu verstehen. Weiterhin habe ich ein Meeting des Developerteams gemacht

um gemeinsam alle Datenstrukturen der Datenbank durchzugehen. Zusammen wurde besprochen was wird gebraucht, was wird ersetzt und was löschen wir komplett raus (Variabel existiert in anderer Form bereits). Zusätzlich mussten Bugs nun über Github getrackt und bei Bearbeitung zugewiesen werden, um Dopplungen zu vermeiden. (https://github.com/users/Gorg-tech/projects/7/views/4) Durch diese Änderungen konnten wir weitere Features besser planen und bestehenden Code leichter anpassen.

Entwicklerdokumentation

Neben meiner Rolle als SCRUM-Master war ich für die Entwicklerdokumentation mit Doxygen verantwortlich. Ich habe die Dokumentation erst im zweiten Semester begonnen. Zu diesem Zeitpunkt war bereits ein Großteil der Implementierung abgeschlossen, was den Nutzen der Dokumentation für die laufende Entwicklung eingeschränkt hat. Besonders für die anfänglichen Verständnisprobleme im Backend wäre eine frühzeitige Dokumentation sinnvoll gewesen. Auch für unseren Teamzuwachs Gabriel wäre eine Dokumentation zur Einarbeitung gut gewesen.

Fazit und Lernerfahrungen

Als SCRUM-Master habe ich dazu beigetragen eine harmonische Teamatmosphäre zu kreieren. Die positive Arbeitsweise hat z.B. die kreative Arbeit des Teams gefördert. Besonders die Entscheidung für Präsenzmeetings hatte einen nachhaltigen positiven Einfluss auf die Zusammenarbeit. Allerdings habe ich gemerkt, dass eine gewisse Struktur für ein größeres Projekt enorm wichtig ist. Wichtig ist eine gute und klare Aufgabenverteilung. Dabei sollten Aufgaben auch an Personen mit weniger Vorwissen vergeben werden, um deren Entwicklung zu fördern und andere Teammitglieder zu entlasten. Gleichzeitig muss jedoch sichergestellt werden, dass die Aufgaben den Fähigkeiten der Personen entsprechen. Hierbei ist es wichtig, dass alle abgeholt und Verständnisfragen geklärt werden.

Die Erfahrungen mit der Entwicklerdokumentation haben mir deutlich gezeigt, dass Dokumentationen kein optionaler Zusatz am Ende des Projektes sind, sondern ein zentraler Bestandteil einer Software. Eine frühzeitige Dokumentation hätte nicht nur Arbeit am Ende gespart, sondern auch Zeit zwischendurch.

Dass die Dokumentation erst spät begonnen wurde, sehe ich rückblickend als verpasste Chance, aus der ich für zukünftige Projekte klar gelernt habe.

Einzelreflexion Vincent Karmanoczki

Rolle und Einordnung in den SE-Prozess

Im Projekt habe ich überwiegend die Rolle eines Entwicklers eingenommen und sowohl im Frontend als auch - im späteren Projektverlauf - im Backend gearbeitet. Meine Hauptaufgaben lagen in den Phasen Entwurf und Implementierung, mit zusätzlichen Berührungspunkten zum Anforderungsmanagement, zur Architektur sowie zur Qualitätssicherung.

Unsere Arbeitsweise folgte grundsätzlich einem iterativen Vorgehen, wobei GitHub Projects zur Aufgaben- und Backlog-Verwaltung genutzt wurde. In frühen Projektphasen war diese Arbeitsweise jedoch noch wenig formalisiert, da es zunächst keine klaren Sprint Plannings gab und Aufgaben häufig erst im Weekly neu verteilt wurden. Dies wirkte sich insbesondere auf die Klarheit von Anforderungen und Schnittstellen aus.

Zu Projektbeginn arbeitete ich primär im Frontend und setzte dort auf Basis der vorhandenen Wireframes konkrete Views um. Im weiteren Verlauf eignete ich mir zusätzlich Kenntnisse im Backend (Django) an, um Abhängigkeiten zwischen Frontend und Backend besser auflösen und Features end-to-end umsetzen zu können. Da ich zu Beginn keine Vorerfahrung mit den eingesetzten Technologien hatte, bestand eine zusätzliche Herausforderung darin, mich parallel in neue Frameworks, Programmiersprachen und Projektstrukturen einzuarbeiten.

Eigene Beiträge und Lösungswege

1. Umsetzung zentraler Frontend-Views im Kontext unklarer Anforderungen

Zu meinen frühen Aufgaben zählte die Implementierung mehrerer Frontend-Views in Vue.js, unter anderem des `EditQuizView` und `EditQuestionView` (Issue #64). Diese Aufgaben waren eng mit noch nicht vollständig geklärten Backend-Schnittstellen und Anforderungen verknüpft.

Da zu diesem Zeitpunkt weder die API-Endpunkte final definiert noch ausreichend dokumentiert waren, fehlte eine klare einheitliche Schnittstelle zwischen Client und Server. Dies erschwerte die Umsetzung funktionaler Anforderungen erheblich, da unklar war, welche Daten persistent gespeichert werden und welche lediglich als temporäre Platzhalter dienen sollten. Infolgedessen mussten die Views im weiteren Projektverlauf mehrfach angepasst werden, etwa beim Routing oder nach Fertigstellung der API-Endpunkte (z. B. Commit f10e83a).

Aus Sicht des Software-Engineering-Prozesses lässt sich diese Phase der frühen Entwurfs- und Implementierungsphase zuordnen, in der der Fokus primär auf einem lauffähigen Produkt lag. Aspekte wie saubere Schnittstellendefinitionen, Dokumentation und Code-Qualität traten zunächst in den Hintergrund. Für mich wurde hier besonders deutlich, wie wichtig frühzeitiges Anforderungsmanagement sowie explizite API-Endpunkte für eine nachhaltige und wartbare

Implementierung sind.

2. Backend-Einarbeitung und Architekturverbesserung

Im weiteren Projektverlauf zeigte sich, dass viele Features ohne Anpassungen im Backend nur eingeschränkt umsetzbar waren. Da im Team insgesamt wenig Erfahrung mit Django vorhanden war, habe ich begonnen, mich selbstständig in das Backend einzuarbeiten, um Abhängigkeiten zwischen Frontend und Backend besser auflösen zu können.

Ein konkretes Beispiel ist ein Fehler bei der Quiz-Erstellung, bei dem Antwortmöglichkeiten nicht korrekt gespeichert wurden (Issue #112). Der initiale Bugfix erfolgte in einem eigenen Branch (bugfix/create-quiz). Im weiteren Verlauf stellte sich jedoch heraus, dass das zugrundeliegende Architekturkonzept - eine separate API-Anfrage pro Frage - nicht performant war. Auf Basis dieser Erkenntnis habe ich die Lösung refaktoriert und die gesamte Quiz-Erstellung in einer einzigen API-Anfrage gebündelt (bugfix/create-quiz-single-api-call).

Dieses Vorgehen lässt sich als inkrementelle Architekturverbesserung im Rahmen iterativer Entwicklung einordnen. Durch die initiale Implementierung wurde ein funktionierendes Feature geschaffen, das anschließend auf Basis von Performance- und Wartbarkeitsanforderungen weiterentwickelt wurde.

Durch diese Einarbeitung konnte ich im weiteren Verlauf zusätzliche Backend-Features übernehmen, unter anderem: - Leaderboard-Funktionalität (Issue #165, feature/streak-leaderboard) - Streak-Tracking (feature/streak-date) - Bewertung von Quiz-Ergebnissen (feature/calculate-iq-points) - Backend-Teil des Freundesystems (feature/backend-friends)

Die technische Umsetzung dieser Features wurde jeweils im Weekly abgestimmt, um sicherzustellen, dass sie mit der bestehenden Client-Server-Architektur vereinbar sind.

3. Refactoring, Code-Qualität und Korrektur früher Architekturentscheidungen

Im späteren Projektverlauf wurde im Team reflektiert, dass sowohl Code-Struktur als auch Dokumentation und Datenmodell an Qualität verloren hatten. In diesem Zusammenhang übernahm ich die Aufgabe, Backend-Code zu dokumentieren und gemäß Clean-Code- und PEP8-Prinzipien zu refactoren (Issue #228).

Dabei zeigte sich, dass zuvor nicht konsequent nach dem YAGNI-Prinzip gearbeitet worden war, was zu ungenutzten Datenbankfeldern und unnötiger Komplexität führte. Diese Altlasten wurden teilweise bereinigt. Parallel erfolgte die Umstellung der Datenbank von SQLite auf MySQL, wobei die Datenbankdatei nicht länger im Repository versioniert wurde - eine wichtige Verbesserung im Bereich Versions- und Konfigurationsmanagement.

Eine weitere zentrale Korrektur betraf eine frühe Architekturentscheidung, bei der dem Client zu stark vertraut wurde. Die fehlende serverseitige Validierung stellte ein Sicherheits- und Fairnessproblem dar, insbesondere im Zusammenhang mit dem Leaderboard. Die nachträgliche

Einführung robuster Validierungsmechanismen (z. B. feature/session-id) war funktional notwendig, jedoch aufwendig, da sie tief in bestehende Logik eingriff.

Zusätzlich wurden UI-Inkonsistenzen sowie veraltete Datenflüsse bereinigt, die aus verworfenen Architekturkonzepten resultierten (bugfix/fill-quiz-overview-fields).

Kritische Bewertung und Lernerfahrungen

Rückblickend hätte ich insbesondere in frühen Projektphasen stärker auf eine explizite Klärung von Anforderungen und Schnittstellen bestehen sollen. Die fehlende Initialdokumentation von API-Endpunkten sowie unklare Task-Abgrenzungen führten zu Mehraufwand und unnötigen Überschneidungen.

Zudem habe ich mehrfach vollständige Features allein umgesetzt, einschließlich Frontend, Backend und API-Logik (z. B. #165, #250, #294). Zwar ermöglichte dies eine geschlossene Umsetzung im Sinne eines vertikalen Feature-Schnitts, stellte jedoch eine hohe persönliche Belastung dar und wäre durch frühzeitigere Abstimmung vermeidbar gewesen.

Positiv hervorzuheben ist meine deutliche fachliche Weiterentwicklung in den Bereichen Backend-Architekturen, Refactoring, Clean Code, Versionsmanagement sowie im Umgang mit iterativen Entwicklungsprozessen. Für zukünftige Projekte nehme ich mir vor, Anforderungen früher zu klären, Aufgaben klarer abzugrenzen und kollaborative Entwicklungspraktiken konsequenter zu nutzen.