Abschlussdokument

-

Zusammenfassung und Rückblick auf das Softwaretechnikpraktikum

Version 1.0

Inhalt

0.1 Zielsetzung des Dokuments

Dieses Dokument gilt als Abschluss des Projekts und dient dazu die wichtigsten Punkte zusammenzufassen sowie ein Resümee zu ziehen. Das Dokument behandelt damit sowohl sachliche und technische Gesichtspunkte als auch ganz individuelle und subjektive Eindrücke. Ziel ist es uns selbst noch einmal den Entwicklungsprozess vor Augen zu führen, um daraus für zukünftige Projekte zu lernen, aber auch unserem Kunden einen Einblick in unsere Arbeitsweise zu geben.

1. Was unsere Projektarbeit ausgemacht hat

1.1 Agiler Entwicklungsprozess – Scrum

Ein wesentlicher Bestandteil unseres Entwicklungsprozesses ist das Scrum-Framework. Gemäß dem agilen Entwicklungsprozess wurden in unserem Team zu Beginn mehrere Rollen verteilt.

Der Scrum-Master nimmt dabei eine **an**leitende Funktion ein. Dabei ist zu betonen, dass er zwar eine leitende Rolle spielt, aber mehr als „Coach“ anzusehen ist. Er behält den Prozess im Überblick und unterstützt bei der Einhaltung. Typischerweise ist der Scrum-Master kein Teil des Entwicklungsteams. Aufgrund des Workloads in unserem Projekt und da das Projektteam wenig Programmiererfahrung, vor allem auch in Java, hat, haben wir uns dazu entschlossen diese Regel aufzuheben und jedes Teammitglied für die Programmierung einzusetzen.

Die zweite klassische Rolle ist der Product-Owner. Er verwaltet das Backlog und koordiniert die Arbeit mit Taiga.

Des Weiteren haben wir die Rollen Testmanager, Werkzeugbeauftrager, Qualitätsmanager und Dokumentationsmanager. Wie die jeweiligen Namen andeuten, setzen die Verantwortlichen mit dem entsprechenden Thema auseinander und stehen für Hilfestellung und Unterstützung bei Fragen bezüglich des Themas zur Verfügung.

Unser Prozess ist Scrum entsprechend in Sprints unterteilt. Wir haben uns für eine Sprint-Länge von einer Woche entschieden, da wir so die wöchentliche Übung nutzen können um mit dem gesamten Team zusammen zu kommen. Diesen Termin nutzen wir um Unklarheiten und wichtige Punkte zu klären sowie zum Abschluss jedes Sprints eine Sprint-Retropersepektive durchzuführen. Bei unserem Studium und der Gruppengröße war es schwierig einen weiteren Termin zu finden, an dem die gesamte Gruppe Zeit hat. So konnten wir zwar die nach Scrum angestrebten Daily-Meetings nicht umsetzen, jedoch war ein regelmäßigeres Treffen (mindestens einmal die Woche unabhängig von der Übung) innerhalb unserer Expertenteams (siehe Abschnitt Expertenteams) durch kleinere Gruppengröße möglich. Zusätzlich dazu standen das ganze Team oder einzelne Expertenteams fast täglich über Slack in Kontakt.

Die regelmäßigen Treffen sorgen dafür, dass Konsenz über das weitere Vorgehen herrscht und alle möglichst immer auf dem aktuellen Projektstand sind, sodass weniger Missverständnisse und Unklarheiten entstehen. Die Sprint-Retroperspektive dient vor allem dazu, festzustellen, was gut gelaufen ist und welche Aspekte noch optimiert werden können. Diese Änderungen können durch die häufigen Treffen sofort in den nächsten Sprint einfließen, was dazu führt, dass der Entwicklungsprozess ständig verbessert und angepasst wird, wodurch ein hoher Qualitätsstandard erreicht werden kann.

Wöchentlich werden weiterhin Stundenzettel abgegeben, welche daraufhin vom Teamleiter ausgewertet werden. Dies ermöglicht es die Arbeitsteilung und den Workload innerhalb der Gruppe besser zu überblicken und ggf. die Aufgabeneinteilung umzustrukturieren und somit zu optimieren.

1.2 Expertenteams

Wir haben uns dazu entschieden neben den oben erwähnten Rollen unser Team weiter zu unterteilen, um effizienter arbeiten zu können. Dabei haben wir unser Team anhand der zu entwickelnden Komponenten unterteilt in zunächst Server-, Desktop-Beobachter und KI-Team. Nach der Messe und dem Einkauf des Android-Beobachters wurden die Teams umverteilt und ein weiteres Android-Teilnehmer-Team gegründet. Bei der Einteilung der Teams haben wir darauf geachtet, dass Kompetenzen und Erfahrungen mit bestimmten Gebieten möglichst gleichverteilt sind. Zur besseren Koordinierung der Teams wurde jeweils noch ein Teamleiter eingesetzt, welcher die Arbeit innerhalb des Teams koordiniert und einen Überblick darüber hat auf welchem Stand der Entwicklungsprozess momentan ist.

Die Einteilung in Expertenteams erlaubt es uns gezielter Aufgaben zu verteilen und konkretere Zuständigkeiten zu erhalten. So können sich die Entwickler auf ihre Aufgabe voll konzentrieren, anstatt von allem ein bisschen zu beherrschen und dazu beizutragen. Dies führt zu effizienterer Arbeit, da jeder ein Experte auf seinem Gebiet wird und sorgt dafür, dass immer klar ist, von wem welcher Code stammt und wen man bei Fragen kontaktieren kann.

1.3 Kommunikation

Bei der Kommunikation innerhalb des Teams haben wir uns einheitlich für das Medium Slack entschieden. Jeglicher Austausch findet hierüber statt. Dabei gibt es sowohl direkte Kommunikation mit einem Gruppenmitglied als auch Channels mit mehreren Mitgliedern für Belange, die ganze Gruppen betreffen. Darunter sind beispielsweise der Channel „allgemein“, ein Channel mit allen Mitgliedern für allgemeine Fragen und Anregungen, sowie der Channel „prog-server“, in dem sich die Mitglieder des Server-Teams befinden, um Aspekte der Server-Programmierung zu besprechen. Die Struktur unserer Gruppe und die Aufgabeneinteilung spiegeln sich so auch in unserer Kommunikation wieder.

Diese Form der Kommunikation war uns sehr wichtig, da man so immer direkt die richtigen Ansprechpartner kontaktiert hat und niemand für einen selbst unrelevante Nachrichten erhalten hat.

1.4 Interne Deadlines

Neben den offiziellen Abgabeterminen haben wir uns innerhalb der Gruppe eigene Deadlines für die Fertigstellung der abzugebenden Ergebnisse gesetzt. Zu Beginn des Projekts wurde uns schnell bewusst, dass dies notwendig ist, da selbst bei einem fertiggestellten Dokument oder Programm noch viele Arbeiten vorgenommen werden müssen, damit es einen dem Kunden gerechten Qualitätsstandard erreicht. Dazu gehören beispielsweise Korrekturlesen, Testen und entsprechende Anpassungen vornehmen. Die internen Deadlines stellen sicher, dass für diese Schritte genügend Zeit bis zur offiziellen Abgabe bleibt.

1.5 Pair- und Group-Programming

Ein weiterer wichtiger Aspekt unserer Entwicklungsarbeit waren das Pair- und Group-Programming. Gerade in der Softwareentwicklung ist Teamwork sehr entscheidend. Oft kommt man nicht weiter oder findet das Problem nicht. Ein anderer Blickwinkel oder eine andere Herangehensweise sind für

die Lösung oft ausschlaggebend. Aus diesem Grund haben wir viel Wert darauf gelegt oft gemeinsam zu programmieren. Entweder persönlich bei einem Treffen oder aber über Medien wie TeamSpeak und Teamviewer. Dadurch hatte man bei Problemen sofort einen oder mehrere Ansprechpartner und konnte so schnell weiterarbeiten, da man nicht lange auf eine Antwort warten musste.

1.6 Einheitlichkeit

In allen Bereichen haben wir großen Wert auf Einheitlichkeit gelegt, um ein möglichst professionelles Auftreten zu erreichen. Dies betrifft sowohl Dokumente und Arbeitsprozesse als auch Code. In diesem Dokument gehen wir nur kurz auf die einzelnen Bereiche ein. Die genauen Merkmale und Richtlinien können Sie in den entsprechenden Guidelines nachlesen.

1.6.1 Programmcode

Bei dem Programmcode war uns Einheitlichkeit wichtig, da es zum einen für ein ordentliches Erscheinungsbild sorgt, aber auch da es zum Besseren Verständnis beiträgt. So erkennt man beispielsweise direkt an der Benennung einer Variablen, wofür sie steht.

1.6.2 Corporate Design – Dokumente

Während unserer Projektarbeit sind zahlreiche Dokumente entstanden. Dazu zählen beispielsweise die wöchentlich entstandenen Protokolle, Abgabedokumente (QA-Dokument, DevOps usw.) oder eigene Richtlinien. Für das Design haben wir uns vor allem an unseren Logo orientiert und somit hauptsächlich mit Blautönen gearbeitet. Für jede Art von Dokument gibt es eine Vorlage, um die Einhaltung des Designs sicherzustellen. Durch das Nutzen einer einheitlichen Gestaltung erreichen wir einen professionellen und ordentlichen Auftritt.

1.6.3 Corporate Design – Benutzeroberflächen

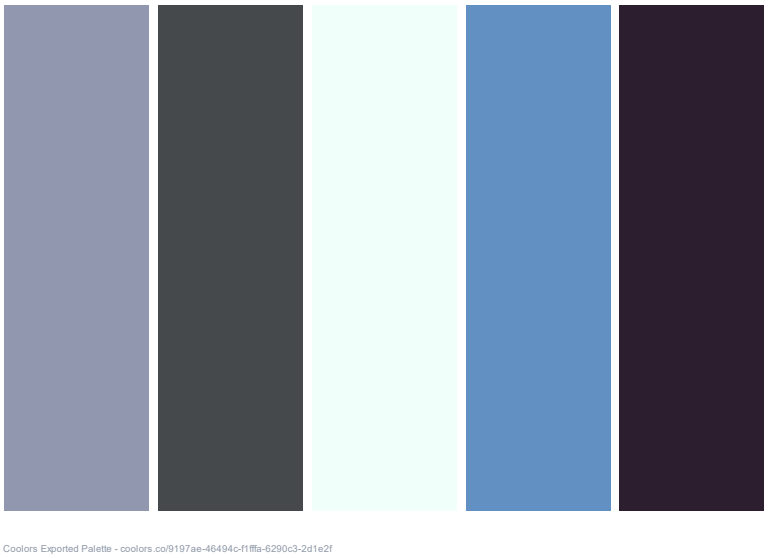

Abbildung 1 - Farbpallete für GUIs

Auch bei den für unsere Kunden entwickelten Produkten haben wir großen Wert auf eine einheitliche Gestaltung gelegt, um den Endnutzern eine möglichst komfortable Nutzung zu ermöglich. Dafür haben wir ein Farbspektrum festgelegt, aus dem die Farben der GUI Elemente gewählt werden.

Zusätzlich zu der Farbauswahl haben wir uns an wissenschaftlichen Erkenntnissen zu unterschiedlichen Bereichen, wie beispielsweise Anordnung und Ausrichtung von Elementen, orientiert.

Dabei haben wir den auf der Messe eingekauften Android-Beobachter weitestgehend von den Richtlinien ausgenommen, da eine vollständige Umstrukturierung sehr zeitintensiv gewesen wäre und unser Fokus auf der Funktionalität lag.

1.6.4 Versionsverwaltung

Da unser Entwicklungsprozess auf Teamwork basiert, kommt es vor, dass mehrere Personen mit den gleichen Dateien arbeiten müssen. Um diese Arbeit zu koordinieren nutzen wir das Versionsverwaltungsprogramm GIT. Auch hier nutzen wir selbst festgelegte Richtlinien um die Arbeit zu strukturieren. So ist immer klar, welche Versionen an welchem Ort zu finden sind und an welchem Ort neue Dateien gespeichert werden.

1.6.5 Testvorgaben

Ein wichtiger Bestandteil des Programmierens ist das Testen. Um sicherzustellen, dass die Software vollständig und korrekt getestet wird, wurde ein Testplan mit Vorgaben für das Testen erstellt. Außerdem wurden Vorlagen für manuelle Tests erstellt. Dieses Vorgehen stellt sicher, dass jeder weiß was und wie er testen muss und von anderen kann besser überblickt werden, was bereits getestet wurde.

1.7 Anpassbarkeit und Erweiterbarkeit

Da im Laufe des Projektes neue Anforderungen auftreten oder bereits bestehende sich ändern konnten, lag unser Fokus bei dem Aufbau und der Strukturierung der Software darauf sie so zu gestalten, dass Änderungen, Anpassungen und Erweiterungen möglichst leicht vorgenommen werden können. Auch im Hinblick auf die Messe und den Verkauf des Beobachters erschien uns dies sinnvoll, da unseren Käufern so die Erweiterung des Beobachters zu einem Teilnehmer erleichtert wird. Um dies umzusetzen haben wir uns für das MVC-Pattern entschieden und es an unsere Komponenten angepasst. Genaueres zu der Architektur der Software finden Sie in dem von uns erstellten DevOps-Dokument.

2. Anpassungen der Definition of Done

Durch die geringe Erfahrung innerhalb unseres Teams, war es schwierig in einem so frühen Stadium des Projektes realistisch umsetzbare Festlegungen für die Definition of Done zu treffen. Dennoch haben wir einige uns sinnvoll erscheinende Vorgaben erarbeiten können. Bis auf kleinere Anpassungen war es uns möglich diese auch umzusetzen.

2.1 Test-Driven-Development

Ein von uns festgelegtes Kriterium für das Abschließen einer Aufgabe war das vollständige Testen des Programmcodes. Zu Beginn haben wir uns für das Vorgehen Test-Driven-Development (TDD) entschieden. Beim TDD werden die Tests für die zu entwickelnde Software vor dem Programm selbst geschrieben. Dies trägt dazu bei, dass die Programmierer ihre Tests ohne Voreingenommenheit anlegen. Viele Fehler in eigenen Arbeiten werden übersehen, da man den Inhalt kennt und weiß was man erwartet. Durch vorheriges Schreiben von Tests wird dieses Problem umgangen. Leider hat sich schnell gezeigt, dass TDD für uns schwer umsetzbar ist, durch Aspekte wie mangelnde Erfahrung im Testen und Abhängigkeiten der Funktionalität untereinander. Der Entwicklungsprozess wurde dadurch verlangsamt, da Programmcode noch nicht geschrieben, bzw. mit anderen geteilt werden konnte, da die zugehörigen Tests fehlten. Aus diesem Grund haben wir uns dazu entschieden, vom TDD abzusehen und die Definition of Done, bzw. den Testplan anzupassen. Das weitere Arbeiten hat schnell gezeigt, dass dies die richtige Entscheidung war und wir wesentlich effizienter arbeiten konnten.

3. Resümee

3.1 Was wir gelernt haben

Innerhalb des Teams wurden verschiedene Erfahrungen gemacht. Sowohl positive als auch negative. Im Folgenden halten wir die wichtigsten Eindrücke fest.

3.1.1 Programmiererfahrung/-Stil

Einige von uns haben schnell festgestellt, dass die Programmiererfahrung für ein solches Projekt noch nicht ausreicht. Dies lag an unterschiedlichen Faktoren. Zum einen gab es Probleme mit der Programmiersprache Java, da mit der neuen Studienordnung die Programmiervorlesungen größtenteils auf Python umgestellt wurden. Andererseits lag es an allgemeiner fehlender Erfahrung. Trotz der Schwierigkeiten gab es für jeden von uns Aufgaben, die wir übernehmen konnten, sodass jeder von uns zum Endergebnis beitragen konnte und dazugelernt hat.

Neben den unterschiedlichen Niveaus kamen auch viele unterschiedliche Programmierstile dazu. Jeder bringt seinen ganz eigenen Einfluss mit in das Projekt, was dazu führt, dass viel Austausch stattfinden musste, um den für die Gruppe richtigen Weg zu finden. Dabei ist zu betonen, dass der Vergleich zwischen den verschiedenen Stilen völlig wertfrei ist, da es nicht grundsätzlich ein richtig oder falsch gibt. Je nach Situation kann das eine oder andere passend sein. Natürlich kommt es dabei auch vor, dass einzelne oder mehrere Personen sich stärker anpassen müssen. Jedoch halten wir diese Erfahrung für sehr wichtig, gerade weil es nicht den einen richtigen Programmierstil gibt. In jeder Gruppe wird man erneut vor dieser Herausforderung stehen und sich gegebenenfalls anpassen müssen, wodurch man selbst auch immer Neues lernen wird.

3.1.2 Teamwork

Wir haben gelernt, dass Teamwork einen sehr großen Bestandteil des Projektes ausmacht. Seien es kleinere Fehler oder grundsätzliches Verständnis, mit der Hilfe von anderen Teammitgliedern kamen wir immer weiter. In vielen Situationen stand man vor dem Problem, dass man den Fehler in seinem Code eigenständig nicht findet. Gerade dann kann ein anderer Blickwinkel auf die eigene Lösung ausschlaggebend sein und eine Antwort auf die Frage liefern.

Aber auch bei Verständnisschwierigkeiten und fehlendem Wissen haben sich andere Teammitglieder sehr viel Zeit genommen zu helfen. Dies hat sowohl uns persönlich als auch das gesamte Projekt weitergebracht.

Ein häufiges Problem war außerdem die Zeit. Konnte jemand eine Aufgabe nicht rechtzeitig fertigstellen, haben sich andere dazu bereit erklärt (Teil-)Aufgaben zu übernehmen.

In vielen Fällen hätte das Fehlen der großen Hilfsbereitschaft große Probleme verursacht und die Arbeit verzögert, aber auch die gesamte Gruppendynamik belastet.

3.2 Was wir anders machen würden

3.2.1 Maven

Zu Beginn des Projekts haben wir uns dazu entschlossen mit Maven zu arbeiten, um TODO. Allerdings hat dies zu vielen Problemen geführt, da immer wieder von Maven verursachte Fehler auftraten, deren Ursache nicht unmittelbar erkennbar war. Diese haben das Entwickeln stellenweise erheblich verzögert und Zeitdruck und Anspannung verursacht.

Im Nachhinein betrachtet hätten wir uns für Gradle entschieden, da wir persönlich damit, während der Arbeit mit dem eingekauften Beobachter, bessere Erfahrungen gemacht und von anderen Teams ähnliche Eindrücke bekommen haben.

3.2.2 Interne Abgaben

Wie in Abschnitt TODO erläutert haben wir uns vor Abgaben interne Deadlines gesetzt, um genügend Zeit für Korrekturen zu haben. Im Großen und Ganzen hat uns dies sehr viel weitergeholfen, allerdings hätten wir es optimieren können. Oft trat der Fall ein, dass trotz eines ungefähren Termins, einige erst kurz vor dem offiziellen Abgabetermin dazu kamen Korrektur zu lesen, was im Endeffekt wieder für ein wenig Zeitdruck gesorgt hat. Man hätte dieses Problem umgehen können, indem man frühzeitig klarere Termine für das Korrekturlesen setzt und sicherstellt, dass alle benötigten Materialien bis dahin vorhanden sind. So würde man beispielsweise festlegen, dass nur Korrekturen bis 5 Stunden vor offizieller Abgabe berücksichtigt werden und alle sich die Zeit nehmen bis dahin Korrektur zu lesen, anstatt den Termin lediglich auf den Tag der Abgabe zu legen.

Auch beim Testen von Code kamen wir öfter in Zeitdruck, da für große Teile der Funktionalität mehrere Komponenten benötigt werden. So konnten viele Aspekte erst spät getestet werden, wenn die andere Komponente noch nicht fertig gestellt war. Auch dies hätte sich mit früheren bzw. härteren Deadlines verhindern lassen.

3.3 Unsere Kritik und Anregungen

3.3.2 Umstellung auf Python

Durch die Umstellung des Studienverlaufs hat sich auch einiges an den Programmiervorlesungen geändert. Während vorher der Fokus auf der Programmiersprache Java lag und diese intensiver behandelt wurde, wird nun vorwiegend Python gelehrt. Das führte dazu, dass viele von uns sich zunächst und auch während des Projektes immer wieder in Java einarbeiten mussten, um arbeiten zu können. Eine intensivere Java-Vorbereitung wäre hier von Vorteil gewesen.

3.3.3 Klausur

Ein weiterer Kritikpunkt von uns ist, dass am Ende noch eine Klausur geschrieben werden muss. Wir fänden es passender und sinnvoller die Abschlusspräsentation als eine Art mündliche Prüfung durchzuführen.

3.3.4 Versteckte Bewertung

In vielen Situationen standen wir vor dem Problem, dass für uns nicht ersichtlich war wie und was bewertet wird. Da alle Dokumente neu für uns waren und keiner Erfahrung mit diesen hatte, fehlte uns durch eine wenig transparente Bewertung oft der Anhaltspunkt, worauf wir achten müssen und hat die Arbeit erschwert.

3.3.5 Werkzeuglast

Durch den theoretischen Charakter des Studiums war wenig bis gar keine Erfahrung mit sämtlichen genutzten Tools innerhalb der Gruppe vorhanden. Fast alle Tools, die wir genutzt haben bzw. nutzen mussten, haben uns meist nur behindert, anstatt uns zu unterstützen. Uns ist bewusst, dass ein Bestandteil des Praktikums eben diese, besonders für das spätere Leben wichtige, Erfahrung ist, allerdings hätten wir uns frühere bzw. bessere und intensivere Einführungen gewünscht, da so die Arbeit sehr erschwert wurde.

Konkret zu GIT möchten wir noch kritisieren, dass es viele Zeiten gab, in denen der GIT-Server nicht verfügbar war. Natürlich sind Wartungsarbeiten etc. Bestandteil eines solchen Systems, allerdings

könnten diese auch zu festen Uhrzeiten durchgeführt werden, sodass man Nicht-Verfügbarkeiten einkalkulieren kann.

3.3.6 Seitenbegrenzungen

Ein Hindernis für uns waren teilweise die Seitenbegrenzungen. Selbstverständlich gilt „Qualität vor Quantität“ und die Begrenzung gibt einem Orientierung, allerdings waren die Begrenzungen unserer Ansicht nach sehr knapp bemessen. Unser Vorschlag wäre, dass jede Gruppe ein Kontingent von X Seiten (bspw. 10) bekommt, welches sie als Extraseiten auf alle Dokumente frei aufteilen kann.