DHBW
Duale Hochschule
Baden-Württemberg
Karlsruhe

Software-Qualität

Hausarbeit

im Rahmen der Prüfung zum

Bachelor of Science (B.Sc.)

des Studienganges

Informatik

an der

Dualen Hochschule Baden-Württemberg Karlsruhe

von

Dominik Ochs

Matrikelnummer, Kurs:	2847475, TINF20B2
Gutachter der Dualen Hochschule:	Dennis Kube & Jonathan Schwarzenböck

Inhaltsverzeichnis

Abbildungsverzeichnis

Quellcodeverzeichnis

1 Einleitung

Softwarequalitätsmanagement (SQM) ist ein entscheidender Aspekt bei der Entwicklung und Implementierung von Softwarelösungen, um sicherzustellen, dass sie effizient, sicher und zuverlässig sind. In dieser Hausarbeit werden wir uns mit dem Vorgehen hinsichtlich SQM auseinandersetzen und Optimierungen entwerfen, indem wir ein praktisches Fallbeispiel untersuchen. Unser Anwendungsbeispiel ist die Entwicklung einer firmeninternen Lernplattform mit Gamification. Dieses Beispiel dient dazu, sowohl theoretisches Wissen als auch praktische Anwendungsfähigkeiten in Bezug auf SQM nachzuweisen.

Bei meiner Firma, der SAP, gibt es intern eine Lernplattform die Gamification nutzt, um SAP-interne Prozesse spielerisch den Kollegen zu vermitteln. Diese Plattform heißt *Digital Heroes*. Bisher können die Kollegen die Lernplattform nach konkreten Inhalten durchsuchen. Außerdem können die Moderatoren bestimmte Inhalte auf der Startseite featuren. Es gibt jedoch keine personalisierten Empfehlungen von Lerninhalten für die Nutzer der Plattform. Abbildung 1.1 zeigt die Landing-Page der Lernplattform.
Ich bin in die Abteilung, um eine in-house Recommendation Engine (RE) zu integrieren, die den Nutzern Inhalte vorschlagen soll. Jedoch besteht die Abteilung zum Großteil aus Studenten, wobei alle Entwickler Studenten sind. Die Lernplattform ist als Studentenprojekt entstanden. Die technischen Komponenten, sowie die Prozesse sind in keinem Zustand, um weitere Features hinzuzufügen. Bevor ich also die Integration der neuen Softwarekomponente durchführen kann, müssen einige grundlegende Änderungen durchgeführt, oder zumindest auf den Weg gebracht werden. Die Verbesserung der bestehenden Codequalität und Entwicklungsprozesse wird Ziel dieser Arbeit sein.

Um dieses Ziel zu erreichen, soll das erworbene Wissen aus der Softwarequalität-Vorlesung genutzt werden, um das Problem kritisch zu analysieren und zu optimieren. Dafür wird zunächst die Applikation und deren Architektur beschrieben, dann die Ist-Situation der Codequalität und der Entwicklungsprozesse untersucht, wobei betrachtet wird, warum die Zustände nicht länger tragbar sind. Daraufhin werden die Gründe erörtert, die zu diesem Zustand geführt haben, sodass nachhaltige Lösungsstrategien entwickelt werden können, weiter werden die relevanten theoretischen Grundlagen aus der Vorlesung dargelegt und schließlich damit erörtert, wie die Soll-Situation aussehen sollte und wie diese erreicht

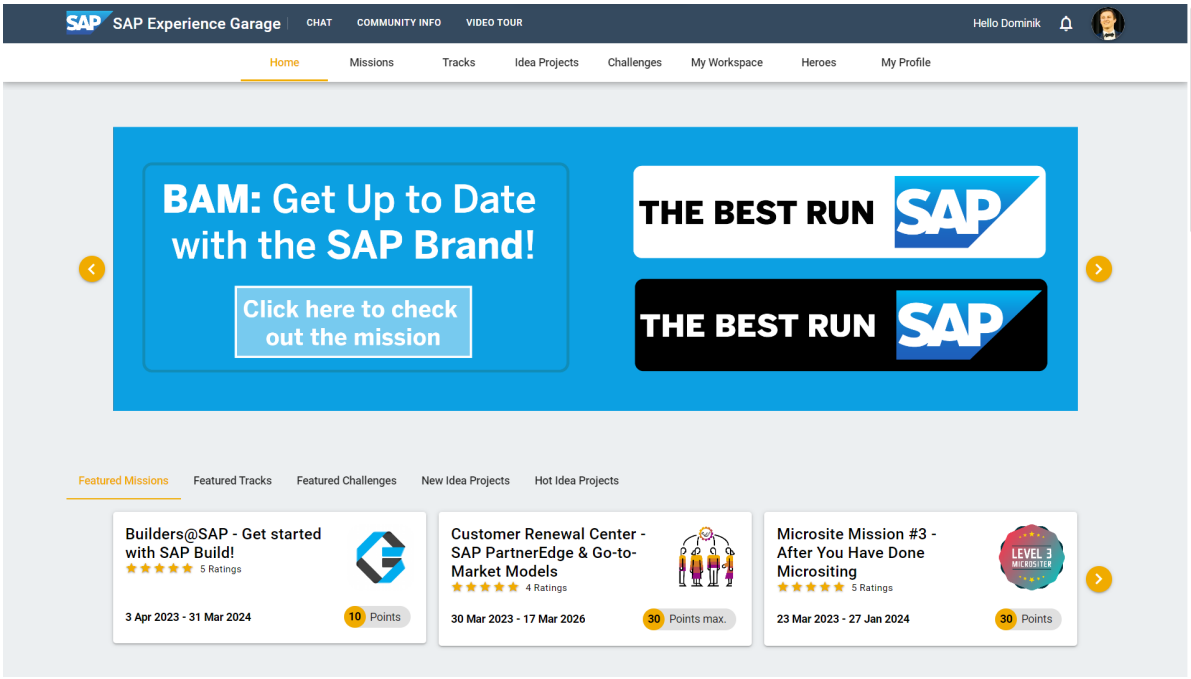

Abbildung 1.1: Die Abbildung zeigt die Landing-Page der Digital-Heroes-Lernplattform. Insbesondere lässt sich der Tab der *Featured Missions* erkennen.

werden kann. Für die Optimierungen werden die Inhalte aus der SQM-Vorlesung herangezogen. Abschließend werden in einer Zusammenfassung die grundlegenden Erkenntnisse und Verbesserungen nochmals bündig dargelegt.

2 Hauptteil

2.1 Architektur der Applikation

Die Architektur der Web-App Digital Heroes mit den verschiedenen verwendeten Technologien lässt sich in Frontend, Backend und DevOps unterteilen und ist in Abbildung 2.1 abgebildet. Es existieren zwei Instanzen der Architektur: ein Entwicklungs- und ein Produktivsystem.

Frontend:

Die Frontend-Anwendung der Digital Heroes Web-App ist als Single Page Application (SPA) konzipiert, die Vue.js und das Quasar Framework verwendet. Vue.js ist ein modernes JavaScript-Framework zum Erstellen von benutzerfreundlichen und reaktiven Webanwendungen. Quasar ist ein weiteres Framework, das auf Vue.js aufbaut und zusätzliche Funktionen und Komponenten bereitstellt, um die Entwicklung von plattformübergreifenden Anwendungen zu erleichtern. Vue.js nutzt die REST-API des Backends.

Backend:

Das Backend der Web-App besteht aus einer Java-Anwendung, die Eclipse Jersey und Apache Tomcat verwendet. Eclipse Jersey ist eine Implementierung der JAX-RS-Spezifikation und ermöglicht die einfache Entwicklung von RESTful Web Services für Servlet Container. Apache Tomcat ist ein Servlet-Container, der als Webserver zum Bereitstellen von Java-Anwendungen dient und das Java-Backend hostet. Als Datenbank kommt die HANA-Datenbank zum Einsatz, die eine high-performance, in-memory relationale Datenbank ist und von SAP entwickelt wird. Das Java-Backend nutzt das Eclipse-Framework und die HANA-Datenbank um eine REST-API für das Vue.js-Frontend bereitzustellen.

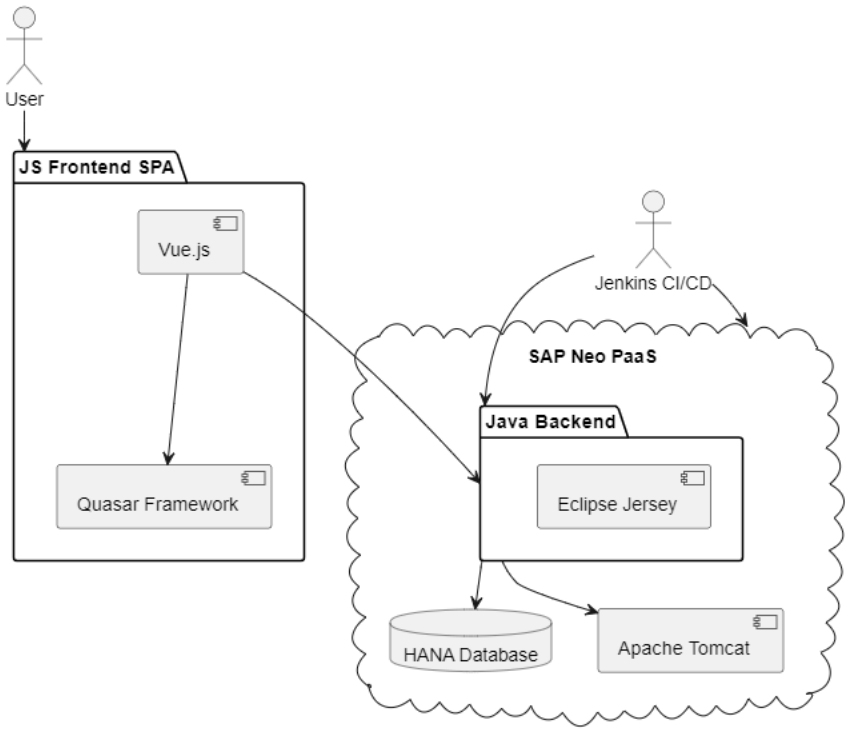

Abbildung 2.1: Die Abbildung zeigt die Architektur der Digital Heroes Lernplattform.

DevOps:

Im Bereich der DevOps-Infrastruktur kommt Jenkins zum Einsatz, eine Open-Source-Automatisierungssoftware, die zur Implementierung von Continuous Integration und Continuous Deployment (CI/CD) verwendet wird. Jenkins ermöglicht die Automatisierung von Build- und Testprozessen sowie die Bereitstellung der Anwendung in einer kontrollierten und konsistenten Weise. Die Web-App wird auf der SAP Neo Platform-as-a-Service (PaaS) Cloud-Plattform gehostet, die eine skalierbare und einfach zu verwaltende Umgebung für die Bereitstellung und den Betrieb der Anwendung bietet. Wird auf den Main-Branch des verwendeten Github-Respositories gepusht, bzw. eine Pull-Request gemerged, startet Jenkins den CI/CD-Prozess und testet und deployed das neue Backend/Frontend auf der Entwicklungsinstanz. Änderungen im Frontend werden zudem direkt auf dem Produktivsystem deployed. Für Änderungen des Backends im Produktivsystem kann in SAP Neo ein neuer Release erzeugt werden, bei dem das Java-Backend der Entwicklungsinstanz auf das Produktivsystem kopiert wird.

2.2 Ist-Situation und warum die problematisch ist

Das Entwickler-Team ist auf Deutschland und Australien aufgeteilt, was zu Kommunikationsschwierigkeiten und unterschiedlichen Arbeitszeiten führt. Dies erschwert die Zusammenarbeit und kann die Qualität der Software und die Effizienz der Entwicklungsprozesse beeinträchtigen.

Es gibt keine fest vorgegebenen Release-Zyklen; Releases finden nur statt, wenn ein neues Feature fertig ist. Dies kann zu unvorhersehbaren und unregelmäßigen Updates führen, wodurch die Planung und das Management des Projekts schwieriger werden. Außerdem werden Features möglicherweise nicht ausreichend vor einem Release getestet.

Releases werden oft freitags durch das australische Team durchgeführt. Dies ist problematisch, weil das Team in Australien möglicherweise nicht verfügbar ist, um eventuell auftretende Probleme sofort zu beheben, was zu längeren Ausfallzeiten führt. Das deutsche Team muss dann freitags die Probleme beheben oder einen Roll-Back veranlassen, wenn die Probleme zu groß sind. Insbesondere muss sich das deutsche Team aber erst in den Code des australischen Teams einlesen, was das Beheben der Fehler viel ineffizienter gestaltet, als wie wenn das australische Team diese Fehler beheben würde.

Es gibt keinerlei automatisierte Tests. Das bedeutet, dass die Qualität der Software nicht kontinuierlich überprüft wird, wodurch Fehler erst spät oder gar nicht entdeckt werden. Dies erhöht das Risiko drastisch, dass Fehler in die Produktion gelangen, was zu einer schlechteren Benutzererfahrung führt.

Viele Bugs gelangen in die Produktionsumgebung, was häufig auf die fehlenden Tests zurückzuführen ist, wodurch oft Hot-Fixes notwendig sind. Dies zeigt, dass der Entwicklungsprozess und die Qualitätssicherung unzureichend sind und die Softwarequalität leidet. Außerdem sind die Hot-Fixes selbst ungetestet und sehr überhastet erstellt worden, was wiederum zu mehr Bugs führen kann. Auch betreffen diese Bugs gelegentlich völlig andere Teile des Systems und fallen nicht sofort auf. Zu allem Überfluss werden diese Hot-Fixes mit diesen gefährlichen Bugs direkt auf das Produktiv-System deployed.

Der CI/CD-Prozess für das Deployment auf der Entwicklungsinstanz dauert 15 Minuten. Das ist eine lange Wartezeit, die den Entwicklungsprozess verlangsamt und zu einer geringeren Produktivität führt. Zur Verschlimmerung der Lage kann das Backend nicht auf dem lokalen Rechner gestartet werden, da die HANA-Datenbank nicht lokal gehostet

werden kann und kein Mock existiert. Dies erschwert die lokale Entwicklung und das Testen, was die Effizienz der Entwickler stark beeinträchtigt und die Qualität der Software gefährdet, da Neuerungen weniger getesetet werden, weil der Testprozess so frustrierend für die Entwickler ist.

Die Codequalität ist in einem schrecklichen Zustand: Viele duplikative Code-Stücke, wenig Struktur und keine Aufteilung der Architektur in Schichten. Es wird immer gegen Implementierungen programmiert, anstatt gegen Abstraktionen. Dies führt zu einer schlechteren Wartbarkeit, schlechterer Lesbarkeit, erschwert das Testen, da Mocks und Fakes nicht möglich sind, erhöht die Fehleranfälligkeit und erschwert die Einarbeitung neuer Teammitglieder, was insbesondere in dem Projekt ein Problem ist, da oft Studenten hier arbeiten, die nicht für lange Zeit eingebunden sind und dadurch hohe Fluktation besteht.
Desweiteren ist der Code ist oft fehleranfällig und unleserlich, zum Beispiel durch 40-50 Zeilen lange, verschachtelte SQL-Statements, die viel Code-Duplikate enthalten. Dies führt zu einer schlechteren Wartbarkeit und macht es schwieriger, Fehler zu erkennen und zu beheben, da oft nicht alle Duplikate angepasst werden.

Es gibt keine Code-Reviews, was bedeutet, dass keine systematische Überprüfung der Codequalität stattfindet. Dies erhöht die Wahrscheinlichkeit von Fehlern und mindert damit die Qualität der Software.

Obwohl ein Kanban-Board mit Tasks und Prioritäten verwendet wird, gibt es kein Sprint-Planning und kein Sprint-Review. Dies führt zu einer schlechteren Planung und Kontrolle der Arbeit und beeinträchtigt die Effizienz des Entwicklungsprozesses, weil keine klarere Absteckung der Aufgaben stattfindet und nicht bewertet wird, wie der vorangegangene Sprint lief.

Die Scrum-Meetings finden zweimal pro Woche statt, aber mit der ganzen Abteilung (nicht nur Entwickler, sondern auch Designer, Manager, Content-Moderatoren usw.). Dadurch werden diese Meetings sehr ineffizient, da nicht alle Teilnehmer direkt am Entwicklungsprozess beteiligt sind. Dies führt zu einer schlechteren Kommunikation und Koordination innerhalb des Entwicklerteams und verlangsamt den Entwicklungsprozess enorm, da die Mitglieder in dem Meeting gebunden sind, obwohl ein Großteil der Inhalte nicht relevant für das jeweilige Mitglied sind. Dadurch wird viel Zeit verschwendet.

Insgesamt sind die identifizierten Probleme im Bereich des Software-Qualitäts-Managements und Entwicklungsprozesse ein erhebliches Hindernis für die Erreichung einer hohen Softwarequalität und einer effizienten Arbeitsweise. Um die Situation zu verbessern, müssen die genannten Probleme adressiert und Lösungen gefunden werden, die zu einer besseren Organisation, Kommunikation und Qualitätssicherung führen.

2.3 Mögliche Ursachen für die identifizierten Probleme

Die geschilderten Probleme in Bezug auf das Software-Qualitäts-Management und den Entwicklungsprozess sind vielfältig und komplex. Es gibt mehrere mögliche Gründe, die zu diesen Schwierigkeiten geführt haben. Um diese Probleme nachhaltig zu lösen, müssen die Gründe verstanden werden, die dazu geführt haben, ansonsten werden nur Symptome behandelt und die Probleme nicht an der Wurzel angegangen. Eine kritische Untersuchung der Situation zeigt folgende mögliche Ursachen:

Unerfahrene Projektleitung: Eine der Hauptursachen für die identifizierten Probleme ist eine unerfahrene oder teilweise auch überforderte Projektleitung. Da die festangestellte Projektleitung wenig Erfahrung im Bereich Softwareentwicklung und Projektmanagement hat, führt dies dazu, dass sie die bestehenden Probleme nicht versteht oder realisiert. Dies führt zu einer unzureichenden Organisation, fehlender Prozesskontrolle und wenig Vertsändnis für Ressourcenaufwand, welcher für Qualitätssicherung eingesetzt wird.

Hoher Anteil unerfahrener Studenten: Die schlechte Qualität der Software und der Entwicklungsprozesse könnte auch auf den hohen Anteil unerfahrener Studenten zurückzuführen sein, die in der Abteilung arbeiten. Da diese Studenten nur für eine begrenzte Zeit im Projekt tätig sind und nicht die erforderliche Erfahrung haben, kann dies zu einer hohen Mitarbeiterfluktuation und einer unzureichenden Kontinuität in der Entwicklung führen. Dies wirkt sich negativ auf die Qualität der Software und die Effizienz der Entwicklungsprozesse aus.

Fehlende Verantwortung und Initiative: Ein weiterer Grund für die identifizierten Probleme könnte sein, dass sich keines der Teammitglieder verantwortlich fühlt, die Probleme zu adressieren und zu verbessern. Dies kann auf mangelnde Kommunikation (insbesondere zwischen deutschem und australischem Team), fehlende Anreize oder eine unklare Rollenverteilung innerhalb des Teams zurückzuführen sein. Wenn niemand das Gefühl hat, für

die Qualität der Software und die Effizienz der Entwicklungsprozesse verantwortlich zu sein, wird es schwierig sein, die Situation nachhaltig zu verbessern.

Kommunikationsschwierigkeiten: Die Aufteilung des Entwicklerteams auf verschiedene Standorte, wie Deutschland und Australien, kann zu Kommunikationsschwierigkeiten führen. Zeitverschiebung, kulturelle Unterschiede und eine eingeschränkte Möglichkeit für persönliche Treffen führen dazu, dass Missverständnisse entstehen und Entscheidungen nicht effektiv getroffen werden.

Hoher Druck durch Nutzeranforderungen: Die Nutzer fordern häufig schnell neue Features, was den Druck auf die Studenten erhöht, diese Anforderungen umzusetzen. Da die Studenten möglicherweise nicht in der Lage sind, "neinßu sagen oder Prioritäten angemessen zu setzen, kann dies zu einer Vernachlässigung von wichtigen Aspekten wie Qualitätssicherung, Testentwicklung und Code-Reviews führen.

Fokus auf Feature-Entwicklung statt Testentwicklung: Die studentischen Entwickler legen den Fokus eher auf die Implementierung neuer Features, anstatt auf das Schreiben von Tests, da das Erstellen neuer Funktionen als ßpannenderöder "produktiver"wahrgenommen wird. Außerdem wird durch die fehlende Erfahrung das Schreiben von Tests und anderen Qualitätssicherungsmaßnahmen als eher unwichtig eingeschätzt.

Mangel an Vollzeitentwicklern: Da es keine Vollzeitentwickler oder Senior Development-Experten gibt, wird die Entwicklungsleitung nicht durch einen erfahrenen Entwickler übernommen, der die nötige Führung für die jungen studentischen Junior Entwickler bieten könnte.

2.4 Grundlagentheorie für Optimierung der Ist-Situation und identifizierte Probleme

In diesem Abschnitt wird die Theorie aus der Softwarequalitäts-Vorlesung dargelegt und erklärt, die für die Optimierungen des gegebenen Fallbeispiels der Digital-Heroes-Plattform notwendig ist. Zudem wird kurz erwähnt, wofür jeder Theoriepunkt später verwendet wird, um den Überblick zu verbessern.

Das magische Dreieck

Zunächst soll das magische Dreieck des Projektmanagements erklärt werden, da es später genutzt wird, um die Priorisierung besser zu bestimmen. Es ist ein grundlegendes Konzept, das die drei wichtigsten Faktoren eines Projekts darstellt: Zeit, Kosten und Umfang. Das magische Dreieck zeigt die Beziehung zwischen diesen Faktoren und wie sie sich gegenseitig beeinflussen.

Zeit: Zeit bezieht sich auf den Zeitrahmen, innerhalb dessen das Projekt abgeschlossen werden muss. Bei Digital Heroes sind das bspw. bis wann neue Features erstellt sein sollen.
Kosten: Kosten beziehen sich auf das Budget und die Ressourcen, die für das Projekt zur Verfügung stehen. Im Anwendungsbeispiel sind das neben dem Budget auch die Entwickler, da für einige der Studenten, wie z.B. mich als dualer Student, die Abteilung nicht bezahlen muss.
Umfang/Scope: Der Umfang oder auch Scope bezieht sich auf die Gesamtheit der Arbeit, die innerhalb des Projekts geleistet werden muss. Bei Digital Heroes wäre das also z.B. die Anzahl und Größe der neuen Features, die gefordert wird. Im Umfang sind aber auch Maßnahmen zur Qualitätssicherung enthalten.

Das magische Dreieck zeigt, dass Änderungen an einem Faktor das gesamte Projekt beeinflussen können. Zum Beispiel kann eine Verzögerung im Zeitplan, z.B. durch Krankheit eines Entwicklers, zu erhöhten Kosten führen, da zusätzliche Arbeitsstunden für die restlichen Entwickler erforderlich sein können. Oder eine Erweiterung des Projektumfangs, z.B. aufwendigere oder mehr Features, kann dazu führen, dass mehr Budget und Ressourcen benötigt werden, um das Projekt innerhalb des Zeitrahmens abzuschließen. Um die Qualität des Projektes sicherzustellen, sollte auf ein ausgeglichenes Verhältnis der drei Faktoren geachtet werden, sodass man sich in der 'Mitte' des magischen Dreiecks bewegt.

SCRUM und PDCA

SCRUM ist ein agiles Projektmanagement-Framework, das zur Verwaltung und Entwicklung von komplexen Projekten verwendet wird. Im Anwendungsbeispiel soll eine bessere Compliance zu SCRUM helfen einige der Probleme zu lösen. Es basiert auf

einem iterativen und inkrementellen Entwicklungsansatz, bei dem das Projekt in kurze, wiederholbare Zyklen, sogenannte Sprints, aufgeteilt wird.

Die wichtigsten Bestandteile von SCRUM neben den Sprints sind:

1. Product Backlog: Eine Liste von Anforderungen und Features, die das Team entwickeln möchte. Im Anwendungsbeispiel liegt das Product Backlog im Kanban Board vor.
2. Sprint Backlog: Eine Untermenge des Product Backlog, die während eines Sprints abgearbeitet werden soll. Bei Digital Heroes nicht wirklich vorhanden.
3. Sprint Planning: Ein Treffen, bei dem das Team den Umfang des nächsten Sprints (Sprint Backlog) festlegt und die Aufgaben, die durchgeführt werden müssen, auf das Team verteilt.
4. Daily Scrum: Ein kurzes Treffen, das regelmäßig (oft täglich) stattfindet, um den Fortschritt des Teams zu besprechen, Hindernisse zu identifizieren und Lösungen zu finden, um die Ziele des Sprints zu erreichen. Bei Digital Heroes zwei mal wöchentlich, jedoch mit undefiniertem, zu großem SCRUM-Team.
5. Sprint Review: Ein Treffen am Ende eines Sprints, bei dem das Team das erreichte Ergebnis den (wichtigsten) Stakeholdern präsentiert und Feedback von diesen erhält. In Digital Heroes nicht explizit vorhanden.
6. Sprint Retrospective: Ein Treffen, bei dem das Team den abgeschlossenen Sprint und den Entwicklungsprozess reflektiert und Verbesserungen für zukünftige Sprints identifiziert. Im Anwendungsbeispiel nicht vorhanden.

Zudem weist SCRUM starke Ähnlichkeit zu dem PDCA-Protokoll (Plan-Do-Check-Act-Protokoll) auf, bei dem eine durchzuführende Handlung geplant (Plan; vgl. Sprint-Planning), dann durchgeführt (Do, vgl. Sprint), bewertet und dabei der Prozess verbessert wird (Ceck, Act; vgl. Sprint Review, Retrospective). SCRUM stellt im Kontext dieser Arbeit also eine leicht abgewandelte Implementation von PDCA dar.

2.5 Optimierungen

Im Folgenden werden konkrete und einschlägige Maßnahmen vorgeschlagen, um unter Berücksichtigung der erörterten vielfältigen Gründe die identifizierten Probleme aus den vorangegangenen Abschnitten zu beheben. Dabei sind Schlüsselmaßnahmen SCRUM ordentlich zu implementieren und das magische Dreieck zu berücksichtigen. Wie das genau Anwendung findet, ist im Weiteren detaillierter dargestellt. Für SCRUM soll jedoch ein zweiwöchentlicher Sprint mit zwei SCRUM-„Dailys“ pro Woche dienstags und donnerstags durchgeführt werden. Zwei Wochen, um relativ kurze Zyklen zu haben, bei denen die in Teilzeit angestellten Studenten jedoch auch etwas erreichen können. Zwei SCRUM-Meetings pro Woche, um den Load an Meetings für die Studenten nicht zu stark zu erhöhen. Es kommen schließlich bereits Sprint-Planning, -Review und -Retrospective dazu.

Kommunikation und Zusammenarbeit im verteilten Team verbessern:

- Regelmäßige, klar strukturierte Videokonferenzen zwischen den beiden Teams. Das wird mithilfe von SCRUM-Dailys (bzw. SCRUM-„bi-weekly“) umgesetzt.
- Einsatz von Kanban-Board verbessern, in dem Aufgaben klarer einer Person zugeteilt werden, im Rahmen des neu eingeführten Sprint-Plannings. Das steckt auch den Umfang für jede Person im Rahmen des magischen Dreiecks ab.
- Verantwortlichkeiten besser verteilen, in dem ein deutscher und australischer Team-Lead ernannt wird.

Fest vorgegebene Release-Zyklen einführen:

- Einführung von regelmäßigen Release-Zyklen im Rahmen der zweiwöchigen Sprints, um Planung und Projektmanagement zu erleichtern und vorhersehbarere Releases für die Nutzer zu generieren.
- Zeit für ausreichendes Testen und Qualitätssicherung in den SCRUM-Sprint einplanen. Das Mehr an Zeit sorgt im magischen Dreieck für weniger Kosten, die durch Bugs später verursacht werden.

Releases außerhalb der kritischen Zeitfenster planen:

- Releases während der gemeinsamen Arbeitszeit beider Teams planen, um schnelle Reaktionen auf Probleme zu ermöglichen.
- Insbesondere: Releases sollten nicht freitags stattfinden.
- Klare Zuständigkeiten für die Behebung von Problemen nach einem Release festlegen. Der Autor eines Features oder Code-Stücks (festgelegt über SCRUM-Planning) ist für dessen Funktionieren und ggf. fixen verantwortlich.

Automatisierte Tests implementieren:

- Test-Driven Development (TDD) einführen, um Qualitätssicherung von Anfang an in den Entwicklungsprozess zu integrieren. Dies erhöht initial die Kosten und den Umfang des Projektes im magischen Dreieck aber senkte später Kosten wegen weniger Bugs.
- Unit-Tests, Integrationstests und End-to-End-Tests erstellen, um die gesamte Funktionalität der Software abzudecken. Auch hier ist wieder ein Zeitaufwand nötig, der langfristig allerdings die Kosten (und den Zeitaufwand) senkt, da weniger Zeit mit dem Fixen komplizierter Bugs vergeudet wird.

CI/CD-Prozess optimieren und lokale Entwicklungsumgebung verbessern:

- Beschleunigung des CI/CD-Prozesses mithilfe von Jenkins durch effizientere Build- und Deployment-Strategien untersuchen.
- Erstellung einer lokalen Entwicklungsumgebung durch Abstraktion von Abhängigkeiten mit Interfaces und Mocks oder leichtgewichtigen Alternativen zur HANA-Datenbank. Auch hierfür ist im Rahmen des magischen Dreiecks ein Zeitaufwand notwendig, der später allerdings zu produktiverer Arbeit führt, da Entwickler nicht lange auf das Durchlaufen der CI/CD-Pipeline warten müssen.

Codequalität verbessern:

- Einführung von Code-Richtlinien und Best Practices für saubere Architektur, Abstraktion und Modularität.
- Jedes zu veröffentlichende Code-Stück, zumindest im Backend, muss Code-Reviews von mindestens einem anderen Entwickler als dem Autor selbst durchlaufen. Feedback zur Verbesserung der Codequalität muss vor dem produktiven Deployen des Codes eingearbeitet werden.
- Refactoring von vorhandenem Code durchführen, um Duplikate, schlechte Struktur, fehleranfällige und unleserliche Passagen zu bereinigen.

SCRUM einsetzen und effizientere Meetings durchführen:

- Einführung von Sprint-Planning, Sprint-Review- und Sprint-Retrospective-Meetings, um Arbeitsaufwand besser zu planen, zu bewerten und den SCRUM-Prozess zu verbessern. Hierbei kommt das Prinzip von PDCA zum Einsatz. Hiermit kann auch die Lage des Projektes im magischen Dreieck besser eingeschätzt werden.
- Scrum-Meetings auf die direkt am Entwicklungsprozess beteiligten Personen beschränken, um Effizienz zu steigern. Dafür in den Sprint-Reviews die weiteren Stakeholder inkludieren und in Feedbackprozesse einbinden.

Ressourcen- und Prioritätenmanagement verbessern:

- Schaffung einer Kultur, in der „nein“ sagen erlaubt ist, unterstützt und encouraged wird und Prioritäten angemessen gesetzt werden können.
- Fokus auf Testentwicklung und Qualitätssicherung, anstatt nur auf Feature-Entwicklung. Hierfür zu Beginn ein festes Zeitkontingent einführen.
- Den Wissenstransfer zwischen gehenden und kommenden Studenten verbessern, aufgrund der hohen Studentenfluktation. Hierfür sollte ein Prozess mit einem expliziten Meeting angesetzt werden, in dem ein kommender Student vom restlichen Team alle Dos and Don’ts, sowie die Erfahrungen der vorhandenen Studenten erzählt bekommt. Insbesondere sollte darauf hingewiesen werden, welche Herausforderungen

zuvor bewältigt werden mussten. Ein gehender Student sollte seine einschlägigsten fünf Erfahrungen mit Bezug zu Softwarequalität für die Nachwelt dokumentieren.

- Auf lange Sicht und wenn das Projekt wächst Bemühungen zur Bereitstellung von Ressourcen für die Anstellung von mindestens einem Senior-Entwickler, der langfristig das Team unterstützen und leiten kann, unternehmen. Natürlich ist es hier einfach zu sagen „einfach mehr Entwickler anstellen“, aber es sollte wirklich im Interesse der Projekt- und Abteilungsleitung sein Ausschau nach zumindest einem Vollzeitentwickler zu halten, wenn die Ressourcen verfügbar werden sollten, anstatt z.B. einen weiteren Vollzeitdesigner anzustellen (wofür z.B. Ressourcen vorhanden sind).

Umgang mit Zeitdruck und Erwartungen der Nutzer:

- Klare Kommunikation gegenüber den Nutzern über realistische Erwartungen bezüglich der Bereitstellung neuer Funktionen. Hierfür kann den Benutzern zum besseren Verständnis das magische Dreieck nahegelegt werden: Die Nutzer beteiligen sich nicht an den Kosten des Projektes, dementsprechend kann kein allzu großer Umfang neuer Features in wenig Zeit erwartet werden, da sonst die Kosten drastisch ansteigen würden.
- Zeit für Qualitätsmanagement, Testentwicklung und -durchführung in den Entwicklungszyklus einplanen, um den Zeitdruck zu reduzieren. Das sollte insbesondere im Sprint-Planning berücksichtigt werden. Für ein Feature sollte also immer auch die Zeit zum Testen und zur Sicherstellung der Code-Qualität mit einberechnet werden.

3 Zusammenfassung