„Prototyp zur Synchronisierung von analogen und digitalen Scrum-Boards“

Praxisprojekt

Alexander Strutz

Betreuer: Prof. Dr. Matthias Böhmer,
Oliver Blum

29. Juli 2019

Inhaltsverzeichnis

1. Einleitung

Laut der Studie "Status Quo Agile 2016/2017" der Hochschule Koblenz verwenden 63% der Software-Firmen und Agenturen agile Methoden in ihren täglichen Arbeitsprozessen [1]. Agile, auch leichtgewichtig genannte, Prozesse können in verschiedenen Frameworks realisiert werden, jedoch orientieren sich alle an dem Rahmenwerk des "Agilen Manifests". Das agile Manifest setzt den Fokus auf die vier Werte "Individuals and interactions over processes and tools", "Working software over comprehensive documentation", "Customer collaboration over contract negotiation" und "Responding to change over following a plan" [2]. Aus diesen Werten ergeben sich zwölf Prinzipien, die dem agilen Manifest angehören. Jedes agile Framework versucht die Werte und Prinzipien des Manifests umzusetzen und praktisch zu integrieren.

Das in 82% der befragten Firmen genutzte Framework "Scrum" gilt als meist genutzte agile Arbeitsmethode [1]. Im Vordergrund stehen hierbei selbstorganisierte Teams. Diese Teams arbeiten interdisziplinär und bestehen aus drei bis neun Mitgliedern [3]. Sie "entscheiden selbst, wie sie ihre Arbeit am besten erledigen, anstatt dieses durch andere Personen außerhalb des Teams vorgegeben zu bekommen" [3]. Die zu erledigenden Aufgaben ("Stories" genannt) werden durch den "Product Owner", einem Mitglied des Teams, in Absprache mit dem Kunden definiert und im "Product Backlog" priorisiert. Das Product Backlog listet alle Stories auf, die vom Product Owner erstellt wurden.

Zu Beginn eines jeden Arbeitsinkrements, auch "Sprint" genannt, wählt das Team gemäß Priorisierung die Stories, die im folgenden Sprint zu bearbeiten sind. Jeder Sprint erhält ebenfalls ein Backlog, in dem Stories des Sprints nach ihrer Priorisierung gelistet enthalten sind.

Die Stories werden anhand von "Story Points" in ihrer Komplexität geschätzt. Hierbei werden Punkte nicht nach Zeitaufwand, sondern nach vergleichbaren Stories vergeben. Möglich sind Punkte innerhalb der Fibonacci-Folge bis 40. Die Schätzung erfolgt gemeinsam durch das Team in einem sogenannten "Refinement". Der Sprint sollte so geplant werden, dass am Sprintende alle gewählten Stories erledigt sind. Eine Story kann hierbei drei Status einnehmen: "To Do", "In Work" oder "Done". Um das Team über die Status der einzelnen Stories informiert zu halten, gibt es täglich das "Daily". Hier berichtet jedes Teammitglied welche Story es am Vortag bearbeitet hat und ob sich der Status verändert hat. Um das Sprint Backlog zu visualisieren werden "Boards" verwendet. Bei diesen Boards kann es sich um Tafeln, Whiteboards oder ähnliches handeln, auf denen die Stories auf Zetteln ausgedruckt aufgehangen werden. Hierbei gibt es drei Spalten, die die Status der Stories abbilden. Im Daily werden die Stories gemäß dem aktuellen Status verschoben. Die Organisation und Administration des Boards obliegt dem "Scrum Master", der "dafür verantwortlich ist, Scrum entsprechend des Scrum Guides zu fördern und zu unterstützen" [3].

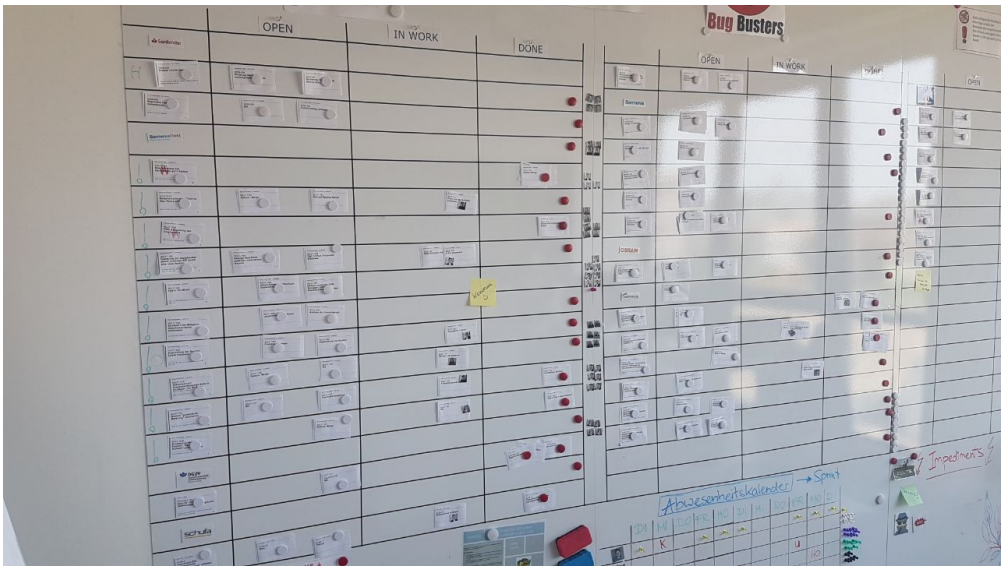

Abbildung 1: Beispiel eines physischen Boards im Unternehmen kernpunkt

Das Board dient dem Team zur übersichtlichen Visualisierung des Sprints. Deshalb sollte es sich räumlich nah am Team befinden, bzw. gut vom Team erreichbar sein. Dem Kunden fehlt die Übersicht am Board jedoch, da dieser sich meistens nicht in direkter Nähe zum Team befindet. Daher wird die Darstellung der Stories oft durch Software wie z.B. "JIRA" von der Firma Atlassian umgesetzt. Auch hier lassen sich Stories in einem Sprint verplanen, mit Punkten schätzen und mit einem Status versehen.

Abbildung 2: Darstellung einer Story in JIRA

Da das Unternehmen "kernpunkt", in welchem diese Arbeit entwickelt wird, JIRA verwendet, fokussiert sich diese Arbeit auf JIRA als digitales Board und die zusätzliche Nutzung von physischen Boards

2. Grundlagen

Im Folgenden wird eine Einleitung in die Domäne, die Problemstellung mit ihren Folgen, sowie den verwendeten Paradigmen gegeben. Anschließend wird aus dieser Recherche eine Forschungsfrage abgeleitet.

2.1. Problemstellung und entstehende Folgen

Durch die Nutzung eines analogen Boards in der Nähe des Teams und eines digitalen Boards zur Sichtbarkeit beim Kunden treten Inkonsistenzen zwischen den Boards auf. Das Umhängen einer Story im Daily kann von einem System wie JIRA nicht erkannt werden. Somit muss der Status hier nachträglich verändert werden. Ebenso sieht es bei Änderungen im JIRA aus. Hierbei kann sich nicht nur der Status, sondern auch der Titel der Story, die Schätzung oder der Bearbeiter ändern. Dadurch muss die Story unter Umständen nicht nur verschoben, sondern durch den Scrum Master komplett neu ausgedruckt werden.

Doch selbst wenn die Boards durch das Team stets synchron gehalten werden, können Probleme aufkommen. So werden Stories auf dem physischen Board nur im Daily verändert, während dies z.B. im JIRA ständig möglich ist. Daher muss man entweder im Daily das physische Board an das digitale Board angleichen oder das digitale auch erst im Daily aktualisieren. Die direkte Aktualisierung des digitalen Boards stellt jedoch einen Konflikt mit dem Scrum Guide dar, da dieser vorgibt, dass das Daily "ein entscheidendes Meeting zur Überprüfung und Anpassung" ist [3]. Die zweite Möglichkeit das digitale Board auch im Daily zu aktualisieren bringt neue Probleme mit sich. So muss sich ein Entwickler bei der Bearbeitung verschiedener Stories stets den aktuellen Status notieren, um im Daily nicht in Gefahr zu kommen etwas zu vergessen. Hierdurch entsteht ein Mehraufwand. Auch verringert so ein Vorgehen die Sichtbarkeit und Aktualität beim Kunden.

Eine direkte Folge der doppelten Boards ist der Mehraufwand, der für den Scrum Master und das Team entsteht. Denn jede Änderung des Status muss sowohl im Daily auf dem physischen, als auch auf dem digitalen Board vorgenommen werden. Jede weitere Änderung bedarf einen Mehraufwand durch den Scrum Master. Da dieser Aufwand nicht auf einen Kunden abrechenbar ist, entsteht hiermit auch ein finanzieller Aufwand seitens des Unternehmens. Durch eine manuelle Synchronisierung können sich Fehler ergeben. So kann ein Teammitglied vergessen eine Änderung an das physische Board zu übertragen, wodurch Inkonsistenzen entstehen können. Daraus kann eine Fehlkommunikation mit dem Kunden entstehen. So könnten Kunden falsche Stände erhalten und fälschlicherweise unzufrieden mit der Leistung des Teams sein.

Außerhalb des zeitlichen und finanziellen Aufwandes steht diese Vorgehensweise in Konflikt mit dem Scrum Guide. Wie in Kapitel 1 beschrieben soll der Fokus auf Interaktion statt auf Tools liegen. Durch die intensive Administration beider Boards werden hierbei Tools in den Vordergrund gestellt, was dem Scrum Guide widerspricht.

2.2. Lösungsansätze

Um das Problem der Synchronisierung zu beseitigen, können verschiedene Lösungsansätze verfolgt werden. Hierzu werden die Vor -und Nachteile der ausschließlichen Nutzung eines analogen oder digitalen Boards diskutiert. Daraufhin wird ein grober Lösungsansatz beschrieben, der die Vorteile beider Ansätze kombiniert.

Die ausschließliche Nutzung eines physischen Boards ist vom Kunden abhängig, da dies seine Anwesenheit im Daily voraussetzt, beispielsweise durch eine Videokonferenz. Dies ist jedoch nur möglich, wenn es sich nicht um eine Multiprojektumgebung handelt. In dem Falle wäre für jeden Kunden ein einzelnes Daily nötig, was den im Scrum Guide angegebenen Zeitaufwand von 15 Minuten übersteigen würde. Auch müsste das physische Boards so gestaltet sein, dass jeder Kunde nur den Teil sieht, der für ihn bestimmt ist, was sich kaum umsetzen lässt. Ein Vorteil des physischen Boards ist die gesteigerte Kommunikation im Daily. Darüber hinaus zeigt eine Studie von PA Müller und DM Oppenheimer aus dem Jahr 2014, dass der Lerneffekt bei händischen Mitschriften deutlich höher ist als bei Mitschriften auf digitalen Endgeräten [4].

Übertragen auf Scrum-Boards bedeutet das, dass Teammitglieder während des Dailys aufmerksamer sind, wenn es vor einem physischen Board stattfindet. Durch die gesteigerte Auffassungsgabe kann das Team nach dem Daily die Arbeit schneller fortsetzen und hat einen besser Überblick über den Stand der Arbeit. Somit ist die Haptik eines physischen Boards ein Vorteil im Hinblick auf die Produktivität und den Fokus des Teams. Darüber hinaus ist der Fortschritt der Arbeit für Teammitglieder greifbarer, wenn sie die Stories händisch verschieben. Das dadurch im Daily präsente Gefühl des Fortschritts steigert die Stimmung und Motivation im Team [5].

Bei der ausschließlichen Nutzung eines digitalen Boards ist eine Anpassung auf verschiedene Kunden ohne weiteres möglich. So bietet Jira beispielweise die Möglichkeit verschiedene Projekte mit verschiedenen Sichtbarkeiten zu erstellen. Ein weiterer Vorteil ist die Archivierung von alten Stories und Sprints. Nach Sprintende werden die Karten der erledigten Stories entsorgt, während sie im Jira vorhanden bleiben, um sich beispielsweise in späteren Sprints über alte Stories zu informieren [6]. Auch bieten digitale Boards eine bessere Speicherung von Zusatzinformationen. So lassen sich auf einer Karte aus Papier nur wenige Informationen speichern, während ein System wie JIRA Stories, Kommentare, Anhänge und Commits aus einer Versionierungssoftware zuordnen kann. Ein Nachteil von digitalen Boards ist die Einschränkung der Kommunikation und Diskussion innerhalb des Teams [7]. Statt den Status der Stories auf dem Board während des Dailys händisch zu verändern kann dies im JIRA am Rechner des jeweiligen Teammitglieds geschehen, was eine Isolation der einzelnen Teammitglieder hervorrufen kann, da sich deren Kommunikation nur noch auf das JIRA beschränken kann.

Die Kombination dieser Boards würde eine Mischung aus Software-System und modifiziertem physischen Board bedeuten. Die Software bleibt hierbei größtenteils unverändert, sodass sich keine zu integrierenden Änderungen ergeben. Das physische Board muss modifiziert werden, sodass es in der Lage ist mit dem digitalen Board zu kommunizieren. Wichtig ist hierbei eine automatische Synchronisation,

um weiteren Aufwand für den Scrum Master zu verhindern (vgl. Problemstellung in Kapitel 2.1). Am effektivsten wäre hierbei eine bidirektionale Synchronisation, die bei Änderungen auf einem Board das jeweils andere Board aktualisiert. Das physische Board darf hierbei seine intuitive Verwendung seitens der Teammitglieder nicht verlieren.

2.3. Paradigmen und Begrifflichkeiten

Im Folgenden werden verschiedene Begriffe und Konzepte erläutert, die Bestandteil dieser Arbeit sind.

Wie bereits in Kapitel 1 erwähnt gibt es verschiedene Softwarelösungen, die sich mit der Realisierung von digitalen Boards beschäftigen. In dieser Arbeit wird ein System entwickelt, welches mit der Software JIRA der Firma Atlassian interagiert. Grund hierfür ist, dass die API von Atlassian alle nötigen Möglichkeiten zur Kommunikation mit dem Board anbietet. Darüber hinaus ist JIRA mit etwas 125000 Nutzern eine weit verbreitete Software, womit der Prototyp dadurch eine größere Zielgruppe hat [8]. Somit ist der Ausdruck "digitales Board" stets auf ein JIRA-System bezogen, welches für agile Arbeitsorganisation verwendet wird.

Auch ist Kapitel 1 zu entnehmen, dass physische Boards aus verschiedenen Materialien bestehen können. In dieser Arbeit liegt der Fokus auf Whiteboards, auf denen die Stories auf Zetteln mit Magneten befestigt werden (siehe hierzu Anhang C).

Der Begriff "Karte" wird in dieser Arbeit für mit Stories bedruckte Zettel, bzw. für die in Kapitel 6.2 beschriebenen digitalen Zettel verwendet.

2.4. Stakeholder

Im Folgenden werden Stakeholder nach ISO 15288:2015 vorgestellt, die einen Anteil, ein Anrecht, einen Anspruch oder ein Interesse an einem neuen System haben [9]. Das größte Interesse am System haben Scrum-Teams. Da das System dafür ausgelegt wird ihre Probleme bzgl. der Asynchronität von Boards zu lösen, ist ihr Interesse am höchsten, da das System ihre Arbeit direkt positiv beeinflussen wird. Sie werden auch die Stakeholder sein, die das größte Interesse am Erwerb eines solchen Systems haben. Hierbei gibt es leichte Unterschiede zwischen Entwicklern und Scrum Mastern. Entwicklern wird damit die Arbeit genommen den Status von Stories doppelt zu pflegen. Scrum Master erhalten durch das System eine größere Unterstützung, da sie nach der Vorbereitung der Boards zu Sprintbeginn keine späteren Änderungen vornehmen müssen und sich somit auch nicht mit kleinen Anfragen zu Boardänderungen befassen müssen. Dieser (insbesondere finanziell) verringerte Aufwand interessiert auch das Management, sowie das Controlling von agilen Unternehmen, da hierdurch Personalkosten gesenkt werden.

Einen Anteil am System haben Unternehmen, die digitale Scrum-Software zur Verfügung stellen, wie z.B. Atlassian. Da ein solches System mit den Schnittstellen der Board-Software kommuniziert, ist somit auch der Hersteller der Software involviert. Dies

kann sich beispielsweise durch Gebühren bei Nutzung der API bemerkbar machen.

IT-Abteilungen von agilen Unternehmen können einen noch größeren Anteil am System haben, da diese oft die Server bereitstellen, auf denen die Software der digitalen Boards installiert ist. Damit sind IT-Abteilungen oft in der Bereitstellung und Wartung der digitalen Boardsoftware involviert und wären es somit auch in der Bereitstellung des Systems, beispielsweise durch Zugriff auf verschiedene Netzwerke.

Ferner können sowohl Personalabteilungen von Unternehmen, als auch Scrum Coaches Interesse am System haben. Hier steht nicht die Minimierung des Arbeitsaufwandes, sondern die Vermarktung und Präsentation innerhalb der Scrum-Community im Vordergrund. Personalabteilungen können mit so einem fortschrittlichen System neue Mitarbeiter anwerben, während Scrum Coaches das System in Weiterbildungen nutzen können.

Abschließend lässt sich sagen, dass es mehrere Stakeholder mit verschiedenen Interessen und Anteilen gibt, die es in der Entwicklung zu berücksichtigen gilt.

2.5. Forschungsfrage und Vorgehensweise

Aus der Problemstellung ergibt sich folgende Forschungsfrage:

"Wie lassen sich analoge und digitale Scrum-Boards automatisch synchronisieren?"

Durch die Implementation eines Prototypen soll diese Frage beantwortet werden. Zuerst werden die Ziele festgelegt, die durch den Prototypen erreicht werden sollen. Daraufhin werden auf dem Markt vorhandene Lösungen im Hinblick auf die definierten Ziele bewertet. Aufbauend auf dieser Marktrecherche werden die Alleinstellungsmerkmale des Systems festgelegt, welche daraufhin durch Anforderungen in Form von User Stories definiert werden. Danach werden die zu verwendenden Technologien bestimmt, sowie eine grundlegende Architektur konstruiert. Auf Basis dessen wird der Prototyp in Form eines Proof of Concept erstellt und dessen Implementation dokumentiert. Zuletzt wird das System anhand der User Stories evaluiert und die Forschungsfrage im Rahmen eines Fazits beantwortet.

3. Zielsetzung

Im Folgenden werden die Ziele für dieses Projekt definiert, die die Problemstellung adressieren. Dabei wird zwischen strategischen, taktischen und operativen Ziele unterschieden. Sowohl die taktischen als auch operativen Ziele können aus einem oder mehreren Zielen der jeweils höheren Stufe abgeleitet werden.

3.1. Strategische Ziele

1. Produktivitätssteigerung der Teammitglieder

Durch den zusätzlichen Verwaltungsaufwand während des Sprints werden Teammitglieder von ihrer Arbeit abgehalten. Dieser Aufwand muss minimiert werden, um das Team produktiver auszurichten.

2. Vermeidung von Mehrkosten

Insbesondere der Scrum Master muss durch das doppelte Board in Vorbereitung auf den Sprint redundante Arbeiten vollbringen. Diese Arbeit muss vermieden werden, um keine Mehrkosten für das Unternehmen entstehen zu lassen.

3. Vermeidung von Inkonsistenzen

Zwei verschiedene Boards sorgen dafür, dass Informationen über Stories und den Sprint an verschiedenen Orten ausgelagert sind. Diese Informationen sollen stets konsistent sein.

3.2. Taktische Ziele

1. Automatische Synchronisation der Boards

Die händische Synchronisation der Boards ist aufwendig und fehleranfällig. Um diesen Prozess zu verbessern, muss ein System vorhanden sein, welches die Boards ohne oder mit wenig Interaktion des Nutzers synchronisiert.

2. Adaption der Funktionalität

Viele Funktionalitäten, die die Software JIRA bietet, sind für agile Teams notwendig. Daher muss ein angepasstes Board möglichst viele Funktionalitäten aus JIRA abbilden können.

3. Beibehaltung der Haptik

Wie in Kapitel 2.2 beschrieben ist eine haptische Bedienung des Boards wichtig. Aus diesem Grund sollte das Team weiterhin in der Lage sein Stories per Hand zu verschieben.

4. Vereinfachung des Dailys

Die Veränderung des Status in einer Story, sowohl im JIRA als auch auf dem physischen Board bedeutet, dass sich Teammitglieder nach dem Daily noch weiter mit organisatorischen Aufgaben beschäftigen müssen. Diese Organisation nach dem Daily soll vermieden werden, sodass das Team schneller weiterarbeiten kann.

3.3. Operative Ziele

1. Kommunikation zwischen analogem und digitalem Board

Um die Boards synchron zu halten, müssen die Boards miteinander kommunizieren. Daher müssen beide Boards eine technische Schnittstelle haben, über die sie Daten austauschen können.

2. Übernommene Darstellung von Stories

Eine unterschiedliche Darstellung von Stories auf den einzelnen Boards schränkt eine intuitive Benutzung ein. Daher sollte die Darstellung der Stories auf beiden Boards ähnlich sein.

3. Automatische Aktualisierung beider Boards

Um Inkonsistenzen zu vermeiden, müssen beide Boards synchron gehalten werden. Zu diesem Zweck soll sich ein Board selbst aktualisieren, sobald es am anderen Board eine Änderung gibt.

4. Einfache Vorbereitung des Sprints

Insbesondere die Vorbereitung der Boards auf den Sprint ist für Scrum Master zeitintensiv. Daher sollte es dem Scrum Master möglich sein beide Boards in einem Arbeitsschritt vorzubereiten.

5. Zuweisung von Teammitgliedern

Zur besseren Übersicht welches Teammitglied an welcher Story arbeitet ist das Zuweisen von Bearbeitern eine wichtige Funktion von JIRA. Daher sollte es Teammitgliedern möglich sein sich auf beiden Boards in einem Schritt einer Story zuzuweisen.

6. Veränderungen des Status

Die wichtigste Aufgabe im Daily ist die Aktualisierung der Status der Stories. Daher müssen Stories auf beiden Boards so verschoben werden können, dass sich der Status auf beiden Boards anpasst.

4. Analyse bisheriger Produkte auf dem Markt und Ableitung von Alleinstellungsmerkmalen

Im Folgenden werden auf dem Markt erhältliche Lösungen beschrieben und hinsichtlich der Problemstellung analysiert. Dabei werden konkurrierende und inhaltlich ähnliche Systeme erläutert und anhand ihrer Vor- und Nachteile eingestuft, insbesondere in Bezug auf folgende in Kapitel 3.2 definierten Ziele:

- Automatische Synchronisation der Boards
- Adaption der Funktionalität
- Beibehaltung der Haptik

4.1. Erhältliche Lösungen auf dem Markt

4.1.1. Physisches Board mit Nutzung eines Kartenscanners

Eine Möglichkeit analoge und physische Boards zu synchronisieren sind sogenannte Kartenscanner [10, 11]. Hierbei werden die ausgedruckten Karten um einen QR-Code ergänzt. Eine Kamera ist auf das physische Board gerichtet und leitet jede Veränderung an das digitale Board weiter. Hierbei werden die QR-Codes zur Identifizierung der einzelnen Stories verwendet. Je nach System sind die Grenzen zwischen den Spalten ebenfalls durch QR-Codes gekennzeichnet. Wenn eine Karte im JIRA verändert wird, wird dies in der Software angezeigt. Die Software kann als JIRA oder als eigene Webapplikation umgesetzt sein.

Positiv: Die Kamera bietet während des Sprints eine direkte Anpassung des digitalen Boards an das physische. Darüber hinaus bleibt die Möglichkeit Karten per Hand zu verschieben. Auch das Anzeigen von Karten mit unterschiedlichen Status hilft Inkonsistenzen zu vermeiden.

Negativ: Eine Fehlerquelle ist hierbei die Kamera. Denn die Erkennung der Karten könnte ausfallen, wenn es z.B. keine ausreichende Beleuchtung gibt. Die QR-Codes nehmen Platz auf den Karten ein ohne einen Mehrwert für das Team darzustellen. Unter Umständen sind somit größere Karten nötig, was wiederum für weniger Platz auf dem Board sorgt. Die Möglichkeit sich eine Story zuzuweisen fehlt auf dem physischen Board. Auch verringert sich der Aufwand der doppelten Sprintvorbereitung nicht, da ebenfalls Karten ausgedruckt werden müssen.

4.1.2. Display mit Abbildung des JIRA

Die einfachste Lösung das physische Board mit JIRA synchron zu halten ist die Nutzung eines Displays. Dieses Display sollte eine gewisse Größe haben, um das Board anzuzeigen. Während des Dailys verschieben die Teammitglieder die Stories im JIRA mittels eines Rechners, der für das Board da ist.

Positiv: Durch die direkte Nutzung von JIRA entfällt die Aufgabe einer Synchronisierung. Außerdem können Teammitglieder, die sich nicht in der Firma befinden,

am Daily teilnehmen und von überall auf das Board zugreifen. Die gesamte Funktionalität von JIRA kann genutzt werden. So können im Daily Bearbeiter zugewiesen, Storypunkte verändert oder Kommentare geschrieben werden.

Negativ: Die Kosten eines Display sind deutlich höher als die eines Whiteboards oder einer Tafel in der gleichen Größe. Darüber hinaus kann das Display nach dem Kauf nicht vergrößert werden, während physische Boards mit zusätzlichen Boards erweitert werden können. Dies kann vor allem bei sich stetig verändernden Teams zu einem Problem werden. Die Nutzung eines Displays sorgt außerdem für eine geringere Aufmerksamkeit seitens des Teams [4]. Die fehlende Haptik senkt die Motivation des Teams [5].

4.1.3. Smartboards mit Abbildung des JIRA

Die Nachteile eines Displays versuchen Smartboards aufzuheben. Smartboards bieten die Möglichkeit JIRA Boards darzustellen und können statt mit der Maus mit Berührungen gesteuert werden [12]. Smartboards besitzen kein eigenes Betriebssystem, sondern fungieren auch als Displays. Somit können auf diesen ebenfalls jegliche Informationen angezeigt werden.

Positiv: Auch hier lassen sich alle Funktionen des JIRA nutzen. Die Synchronisierung entfällt auch hier. Zusätzlich kann man von einer "Teilhaptik" sprechen, da man das Display mit den Händen bedienen kann.

Negativ: Kosten und Erweiterbarkeit sind ebenfalls ein Nachteil von Smartboards. Auch wird die Konzentrationsfähigkeit im Daily dadruch nicht gesteigert [4]. Auch brauchen die Teammitglieder eine gewisse Einarbeitungszeit, um das Smartboard fehlerfrei und effektiv zu bedienen. Das Fehlen einer speziellen Scrum-Software für Smartboards erschwert die Verwendung als analoges Board.

4.2. Herleiten von Alleinstellungsmerkmalen

Aus der Marktrecherche lassen sich Alleinstellungsmerkmale ableiten, die die Stärken der bereits vorhandenen Produkte mit den zu erreichenden Zielen kombinieren und das System so zu einer attraktiven Alternative gestalten.

4.2.1. Erkenntnisse der Marktrecherche

Die bisher vorhandenen Lösungen und Produkte versuchen meist ein Abbild des JIRAs in ein physisches Board zu übertragen, indem Displays oder ähnliche elektronische Anzeigen verwendet werden. Obwohl man damit ein Board für Dailies und den Überblick im Team besitzt, handelt es sich nicht mehr um ein physisches Board. Viel mehr wird hier das digitale Board dupliziert. Zwar entfällt hierdurch eine Synchronisation, doch auch die in Kapitel 2.2 erwähnten positiven Aspekte des physischen Boards entfallen. Ein angemessenerer Ansatz ist die Nutzung einer Schnittstelle zwischen den Boards. Im Beispiel der Kartenscanner sind dies die QR-Codes, die Kamera und das System, welches mit dem JIRA kommuniziert. Jedoch ist diese Lösung im Vergleich zu den Displays eher umständlich, was die beschränkte Funktionalität und den Aufwand zu Sprintbeginn angeht.

4.2.2. Bidirektionale Aktualisierung

Ein wichtiger Aspekt ist die Möglichkeit das Board in beide Richtungen zu aktualisieren. Das bedeutet, dass auch Änderungen am digitalen Board während des Sprints durch das System an das physische Board übertragen werden, insofern das möglich ist. Somit sind Änderungen am Namen, dem Bearbeiter oder den Storypunkten gemeint. Änderungen des Status werden weiterhin händisch am physischen Board verschoben. Allerdings kann das System den Nutzer hier durch Hinweise über Stories in falschen Spalten informieren. Ebenfalls kann das physische Board bei Veränderungen diese direkt an das JIRA weiterleiten.

4.2.3. Haptik am Board

Ein weiteres Alleinstellungsmerkmal ist die Haptik am Board. Die Stories befinden sich auf Karten am physischen Board, können bewegt und in die Hand genommen werden. Die einzelnen Karten haben eine angemessene Größe. So ist es möglich im Daily Karten zu bewegen und in die Hand zu nehmen, während man über sie spricht.

4.2.4. Vereinfachte Sprintvorbereitung

Bisher müssen Scrum Master zwei Boards für den Sprint vorbereiten, was insbesondere am physischen Board durch Ausdrucken und Aufhängen der Karten zeitintensiv ist. Dieser Aufwand kann vermieden werden, indem der Scrum Master bei der Sprintvorbereitung im JIRA auch direkt das physische Board vorbereiten kann. Dadurch wird der Sprintbeginn vereinfacht, da der Sprint nur noch im JIRA gestartet werden muss und am physischen Board keine weiteren Maßnahmen nötig sind.

4.2.5. Adaptierte JIRA-Funktionalitäten

Bisher sind physische Boards in ihren Funktionalitäten sehr beschränkt, insbesondere im Vergleich zu JIRA. Das Team kann unterstützt werden, indem häufig getätigte Interaktionen am digitalen Board auch am physischen Board verfügbar sind. Hilfreich ist hier die Zuweisung des Bearbeiters, da sich dieser oft im Daily ergibt, wenn die aktuellen Aufgaben für den Tag festgelegt werden.

5. Definition von Anforderungen an das System in Form von User Stories

Im Folgenden werden aus den Alleinstellungsmerkmalen Anforderungen definiert. Diese werden als User Stories nach dem Template von Mike Cohn beschrieben [13]. Die User Stories sind die Grundlage zur Diskussion der Evaluation in Kapitel 9.

Stories und Unteraufgaben auf einer Karte anzeigen
Als Teammitglied möchte ich Stories und Unteraufgaben auf Karten angezeigt bekommen, um am physischen Board den aktuellen Stand des Sprints zu sehen.

Akzeptanzkriterien

- Jede Karte zeigt eine Story oder Unteraufgabe an
- Die Karte zeigt Titel, Abkürzung und Bearbeiter an
- Die Karte aktualisiert sich selbstständig in einem angemessenen Intervall

Story einer Karte zuweisen
Als Scrum Master möchte ich einer digitalen Karte eine Story aus dem JIRA zuweisen, um den Sprint vorzubereiten und ein Abbild des digitalen Boards auf dem physischen Board zu sehen.

Akzeptanzkriterien

- Jede Karte besitzt eine eindeutige ID
- Das JIRA bietet eine Möglichkeit die ID einer Karte einer Story zuzuweisen
- Das System speichert die Zuordnung zwischen Story und Karte

Stories und Unteraufgaben einem Teammitglied zuweisen
Als Mitglied des Teams möchte ich mir im Daily Stories und Unteraufgaben zuweisen, um meine Arbeit und Beteiligung transparent zu halten.

Akzeptanzkriterien

- Ein Teammitglied kann sich im Daily einer Unteraufgabe oder Story zuweisen
- Das System bietet jedem Mitglied eine Möglichkeit sich zu authentifizieren
- Die Karte leitet den neuen Bearbeiter an das JIRA weiter
- Der neue Bearbeiter wird auf der Karte angezeigt

Status von Stories und Unteraufgaben verändern
Als Teammitglied möchte ich den Status von Stories und Unteraufgaben im Daily verändern, um den Stand im Sprint und den Fortschritt transparent zu halten.

Akzeptanzkriterien

- Der Status von Stories und Unteraufgaben lässt sich verändern
- Die Karte leitet den neuen Status an das JIRA weiter

6. Erarbeitung einer Client/Server-Architektur

Im Folgenden wird der Aufbau und die grobe Struktur des Systems inklusive seiner Komponenten beschrieben. Hierbei liegt der Fokus nicht auf einzelnen Technologien, sondern auf die Verteiltheit des System, sowie der Kommunikation zwischen den einzelnen Elementen.

6.1. Anpassungen am physischen Board

Das Board bildet die Basis des Systems und sollte stationär sein. Auf dem Board werden wie bisher Karten platziert, vorzugsweise durch Magneten. Wie bisher soll das Board nicht-digital sein. Dies ist insbesondere für bereits montierte Boards von Vorteil, da sonst eine Aufrüstung mit Aufwand und Kosten verbunden wäre. Außerdem steigert dies die Portabilität des Systems. Somit besitzt das Board auch keine technische Schnittstelle oder Kommunikation mit anderen Komponenten des Systems. Denkbar ist jedoch eine "Aufrüstung" zur Lokalisierung einer Karte. Hierbei sollte ein bereits montiertes Board jedoch nicht umgerüstet, sondern lediglich ergänzt werden. Siehe dazu auch Kapitel 7.4.

6.2. Anwendungslogik auf der digitalen Karte

Die Karte ist eine Repräsentation einer Story oder einer Unteraufgabe. Auf bisherigen Boards wurde dies durch ausgedruckte oder beschriebene Zettel gelöst. Diese Zettel sollen durch eine digitale Lösung ersetzt werden. Dazu wird ein Mikrocontroller oder Mikrocomputer genutzt in Kombination mit einem Display in der Größe einer Papierkarte. Gegebenenfalls wird die Karte mit Sensoren oder Eingabemöglichkeiten ausgestattet, um den Status der Karte zu ermitteln. Wichtig sind hierbei möglichst kleine Abmessungen, damit eine Story nicht mehr Platz wie bisher auf dem Board einnimmt. Auch ein geringes Gewicht ist relevant, damit die Karte mit möglich wenig Magneten hält und nicht der Gefahr eines Sturzschadens ausgesetzt ist.

Aufgrund der begrenzten Speicherkapazität von Mikrocontrollern sollen diese möglichst wenig Software und Anwendungslogik beinhalten. Der Großteil soll auf einen Server ausgelagert werden, da es einfacher ist die Kapazitäten eines Servers als die eines Mikrocontrollers zu erweitern. Unabdingbar jedoch ist das Auslesen der Sensoren, die Ansteuerung des Displays sowie die Kommunikation mit dem Server.

Die Sensorendaten werden ohne eine spezielle Bearbeitung als Rohdaten an den Server weitergeleitet. So werden das Konvertieren, Casten und Filtern der relevanten Daten ausgelagert. Je nach Sensor kann es sich um numerische Daten, wie beispielsweise die ID eines RFID-Tag oder boolsche Werte, wie die eines Tasters handeln. Eine solche Weiterleitung soll nicht nur mit eingehenden Daten, sondern auch mit ausgehenden Daten geschehen, also solche, die zur Ausgabe auf der Karte verwendet werden.

Hierzu soll die Karte eine möglichst universelle Funktion bereitstellen, um Stories und Unteraufgaben anzeigen zu lassen, die sie übergeben bekommt. Das Layout soll

kontinuierlich gleichbleiben, um das Board für das Team konsistent verständlich zu halten. Denkbar ist hier eine Anlehnung an das Layout von JIRA, um die Karte möglichst intuitiv zu halten. Besonders zu beachten ist die maximale Zeichenlänge. Hier sollte für zu lange Texte eine Lösung gefunden werden, bei der der Inhalt der Story oder Unteraufgabe auch gekürzt noch verständlich ist. Dies kann durch eine automatische Abkürzung von bestimmten Begriffen geschehen. Dabei können ausschließlich Abkürzungen verwendet werden, die allgemein bekannt sind, um Missverständnisse zu vermeiden. Auch die Behandlung von Sonderzeichen muss hier beachtet werden.

Die dritte Kernfunktionalität ist die Verbindung mit dem Server. Die Karte muss in der Lage sein sowohl Sensorendaten zu versenden als auch Anweisungen an das Display zu empfangen. Diese Daten werden mit dem Server per HTTP als JSON verschickt und auch als solche empfangen. Näheres zur Verbindung wird in Kapitel 7.5 beschrieben.

6.3. Entwurf eines Anmeldeterminals zur Nutzung in einem Daily

Um sich im Daily einer Story zuweisen zu können ist es notwendig, dass jedes Teammitglied vom Board erkannt und unterschieden werden kann. Dazu wird ein Anmeldeterminal entwickelt, welches ebenfalls aus einem Mikrocontroller besteht. Dieses Terminal soll unabhängig von den Karten sein, sodass diese so klein wie möglich bleiben. Im Gegensatz zu den Karten soll es einem Board zugehörig und fest montiert sein.

Zur Identifizierung der Teammitglieder werden RFID-Tags verwendet. Jedes Mitglied erhält einen Tag, der eine eindeutige ID enthält. Die Zuordnung von IDs zu Mitgliedern ist auf dem Server gespeichert. Damit dieser Tag vom Terminal verarbeitet werden kann enthält es einen entsprechenden Sensor. Wie auch bei der Karte soll ein möglichst großer Teil der Anwendungslogik auf den Server ausgelagert werden. Daher liest das Terminal lediglich die IDs aus und sendet sie zusammen mit der aktuellen Uhrzeit an den Server.

Wird der erste RFID-Tag an den Leser gehalten, beginnt für das System eine Session, also ein Daily mit der ersten angemeldeten Person. Werden nun Karten verschoben, werden sie im JIRA der angemeldeten Person zugewiesen. Sobald der nächste Tag an das Terminal gehalten wird, verändert sich auch der zuzuweisende Bearbeiter der Story. Damit nicht der letzte Teilnehmer des Daily bis zum nächsten Daily "angemeldet" bleibt, wird er nach einer Inaktivität von 10 Minuten abgemeldet.

Alternativ zu einem Anmeldeterminal wäre es auch möglich die Karten mit einem RFID-Sensor auszustatten. Jedoch müssen die Teammitglieder dann ihren Tag an jede Karte einzeln halten, um sich einer Story zuzuweisen, was im Daily störender ist als sich nur zu Beginn anzumelden. Außerdem bekämen die Karten dadurch zum einen mehr Anwendungslogik, aber auch ein größeres Gewicht, was wie zuvor erwähnt vermieden werden soll.

6.4. Skizzierung der Systemarchitektur

In Abbildung 3 wird die verteilte Systemarchitektur erläutert. Hierbei wird das Anwendungsschichtprotokoll HTTP mit der Datenstruktur JSON genutzt. Als Schnittstelle wird hierbei das Konzept REST verwendet.

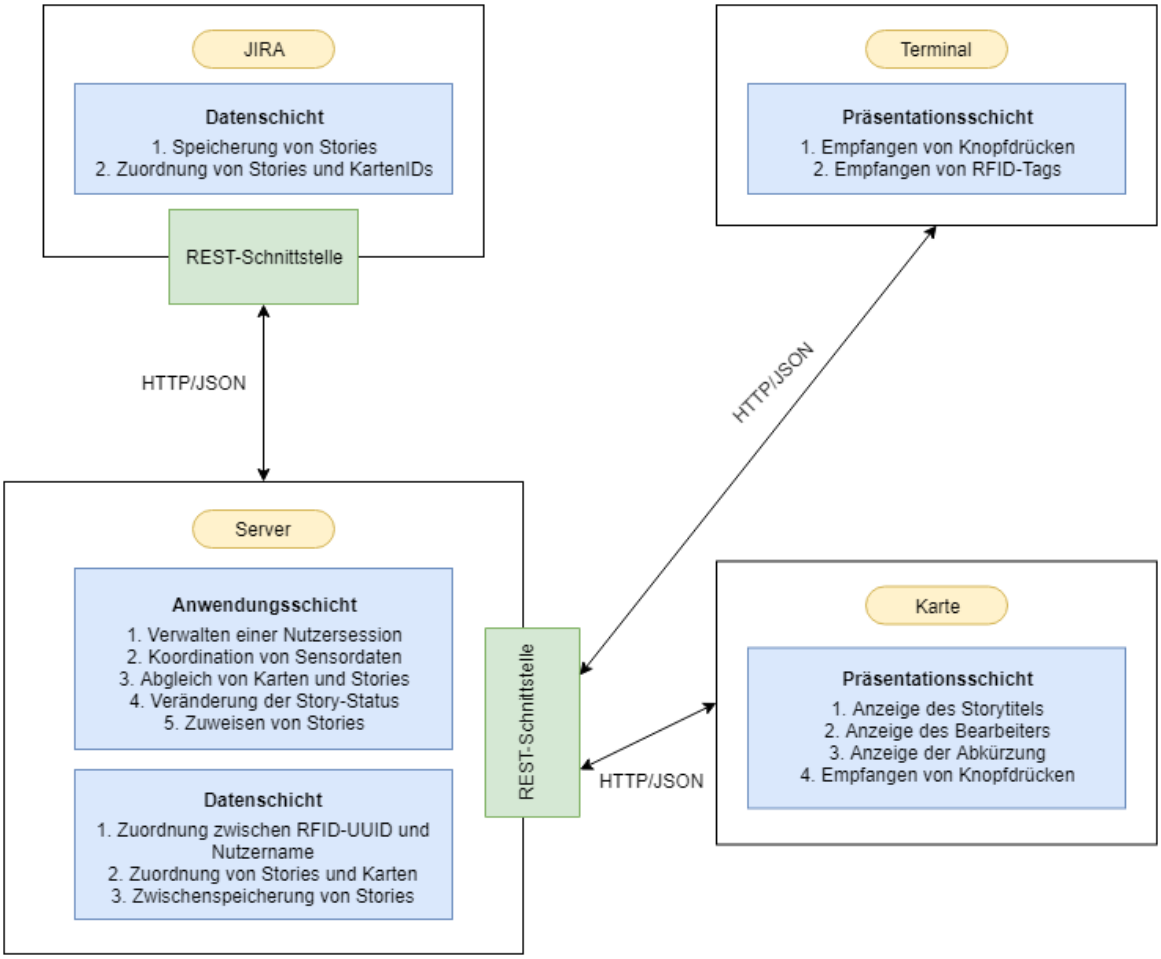

Abbildung 3: Visualisierung der Systemarchitektur

7. Abwägung und Entscheidung zwischen erhältlichen Technologien

In diesem Kapitel werden Technologien für die in der Systemarchitektur enthaltenen Komponenten recherchiert und bewertet. Der Fokus liegt hier auf der Eignung zur Implementation der Anforderungen und Umsetzung der User Stories.

7.1. Mikrocomputer und Mikrocontroller

Jede Karte sowie das Anmeldeterminal benötigen einen eigenen Mikrocontroller. Dieser sollte möglichst klein und leicht sein, um mit wenigen Magneten am Board befestigt zu werden. Für einen Einsatz der Karte über einen Sprint hinweg sollte der Mikrocontroller einen geringen Stromverbrauch haben, sodass er durch einen austauschbaren Akku betrieben werden kann. Von Vorteil ist ein integrierter WLAN-Chip, um Verkabelungen und zusätzliche Hardware zu reduzieren. Weniger relevant sind die Speichergröße und die Rechenleistung, da die meiste Logik auf dem Server hinterlegt werden kann.

7.1.1. VoCore2

Gestartet als Hobbyprojekt brachte der Single-Board-Computer VoCore insgesamt 102.641 € Investitionen über Fundraising ein [14]. Der Chip zeichnet sich insbesondere durch seine Größe aus. Mit seinen Maßen von 25,8mm * 25,8mm ist er kaum größer als ein Centstück [15]. Trotz der schmalen Abmessungen besitzt er einen verbauten WLAN-Chip und 28 generelle Input/Output-Kontakte (GPIO). Auch der Stromverbrauch ist mit 200mA bei Nutzung mit WLAN über 5V gering [16]. Das Format des Chips bringt aber auch Nachteile mit sich. So ist eine Stromversorgung nur über die Kontakte möglich, ein USB-Port ist nicht vorhanden. Das Betriebssystem ist die Linux Distribution OpenWRT. Dadurch ist das Kompilieren von Software lediglich mit Ubuntu Server beispielsweise auf einer virtuellen Maschine möglich, was die Entwicklung des Systems erschwert [17].

7.1.2. Raspberry Pi Zero W

Etwa doppelt so groß wie der VoCore2 ist der Single-Board-Computer Raspberry Pi Zero W der Raspberry Pi Foundation [18]. Mit 512MB RAM Speicher und einem mit 1 GHz getakteten Prozessor bietet er genügend Ressourcen, um ein Betriebsystem und verschiedene Programme parallel laufen zu lassen. Er bietet mit einem SD-Kartenslot, Micro-USB- und HDMI-Anschluss, sowie integriertem WLAN-Modul mehrere Schnittstellen an. Doch dies ist problematisch, da er dadurch einen höheren Stromverbrauch hat. Darüber hinaus sind viele Schnittstellen und Funktionen des Raspberry Pi Zero W für das Projekt nicht nötig.

7.1.3. Arduino UNO

Der Arduino UNO ist ein Single-Board-Mikrocontroller der Firma Arduino. Im Gegensatz zum Raspberry PI ist er mit 16MHz Prozessortaktung und 2 kB SRAM-Speicher deutlich leistungsschwächer [19]. Es ist nicht möglich ein Betriebssystem zu installieren, stattdessen ist der Arduino UNO für simplere Programme wie das

Auslesen von Sensoren geeignet. Das spiegelt sich auch im Stromverbrauch wieder. Dieser lässt sich vorallem durch den "Deep Sleep Mode" reduzieren. In diesem Modus werden ungenutzte Module und Schnittstellen deaktiviert, bis eine definierte Unterbrechung (z.B. Signale eines Sensors) eintritt [20], was für einen dauerhaften Betrieb der Karte am Board von Vorteil ist. Der größte Nachteil des Arduino UNO ist das Fehlen eines integrierten WLAN Moduls. Zwar ist das Nachrüsten mit einem sogenannten Shield möglich, jedoch belegt dieses Kontakte des Mikrocontrollers, die für RFID-Leser und Display benötigt werden. Daher ist auch dieser Controller nicht geeignet.

7.1.4. ESP8266 mit NodeMCU

Eine Kombination der bisher genannten Vorteile ist der ESP8266 Mikrocontroller mit NodeMCU. Auf diesem Chip läuft ebenfalls Arduino, wodurch er für kleinere Programme geeignet ist. Er enthält sowohl ein integriertes WLAN-Modul als auch einen USB-Anschluss. Wie der Arduino UNO bietet auch der ESP8266 die Möglichkeit den Controller in den Deep Sleep Modus zu schalten, in welchem er nur 0,02 mAh verbraucht [21]. Mit seinen Abmessungen von 58mm * 31mm ist er etwa zu vergleichem mit dem Raspberry Pi Zero W. Durch die open-source Firmware NodeMCU lässt sich der Controller mit vergleichsweise wenig Aufwand mit einem REST-Server verbinden [22]. Der größte Nachteil des ESP8266 ist seine verhältnismäßig geringe Zahl an I/O Kontakten. So gibt es nur 16 I/O-Pins. Dieses Problem lässt sich mit einem sogenannten I/O Extender, wie dem MCP23017 lösen. Dieses Bauteil verwendet zwei GPIO-Pins und erweitert diese zu 16. Darüber hinaus kann auch dieser Extender Interrupts empfangen, um den Deep Sleep Modus zu deaktivieren.

Aufgrund des geringen Stromverbrauchs, des Deep Sleep Modus, der Integration der NodeMCU-Firmware und des eingebauten WLAN-Moduls wird der ESP8266 mit NodeMCU, falls nötig in Kombination mit einem GPIO-Extender, verwendet.

7.2. Displays

Die wichtigste Ergänzung des Mikrocontrollers ist das Display. Es soll möglichst lange die selben Informationen anzeigen und verändert sein Bild sehr selten, verglichen mit PC-Monitoren. Der wichtigste Aspekt ist der Stromverbrauch und die Kompatibilität mit dem Deep Sleep Modus des ESP8266. Die Größe des Displays sollte so sein, dass man die Informationen der Story gut lesen kann, die Karte aber nicht zu viel Platz auf dem Board einnimmt. Eine für das Display passende Arduino-Library ist obligatorisch für die Nutzbarkeit.

7.2.1. HD44780 LCD

Das ursprünglich von der Firma Hitachi hergestellte HD44780 LCD-Modul ist in der Lage je 16 Zeichen in zwei Spalten einfarbig anzuzeigen. Hierbei ist eine weiße Schrift auf blauen Hintergrund zu sehen. Die Beleuchtung erfolgt durch weiße LEDs bei einer Auflösung von 16 * 80 Pixel [23]. Die Zeichenkodierung ist auf das Alphabet und wesentliche Satzzeichen beschränkt. Dieses Display eignet sich nicht, da es lediglich 32 Zeichen anzeigen kann und keine Funktion bietet den Text zu formatieren, was

zum einen die Möglichkeiten der Anzeige einschränkt als auch das in Kapitel 3.3 beschriebene zweite Ziel verfehlt.

7.2.2. OLED Display

Im Gegensatz zum eingeschränkten LCD ist es beim OLED möglich das Display pixelgenau anzusteuern. Somit ist außerhalb von Texten auch die Anzeige von Bildern und Grafiken möglich. Dies ist notwendig, um auch Storypoints und Bearbeiter, sowie Warnungen zum falschen Status darzustellen. Im alltäglichen Gebrauch kann die Stromversorgung ein Problem darstellen. So braucht das Display im Betrieb eine dauerhafte Stromzufuhr. Für einen Prototypen wäre das unproblematisch, allerdings wäre ein OLED nicht für eine permanente Nutzung geeignet, da auch der Deep Sleep nicht verwendet werden könnte.

7.2.3. eInk Display

e-Ink Display haben die Besonderheit keine dauerhafte Stromzufuhr zu benötigen. Das ist ein massiver Vorteil gegenüber den bisher vorgestellten Anzeigemöglichkeiten. Ein e-ink Display enthält Mikrokapseln, die mit schwarzen (negativ geladenen) und weißen (positiv geladenen) Partikeln gefüllt sind. Die Kapseln befinden sich in einer konduktiver Flüssigkeit. Durch Anlegen von elektrischer Spannung verändern die Kapseln ihre Ladung, wodurch sich ihre Farbe ändert, die sie auch nach Beendung der Stromzufuhr behalten [24]. Daher eignen sich e-Ink Displays sehr gut für eine längere Nutzung, insbesondere auch für die Nutzung im Deep Sleep Mode, was die Akkulaufzeit einer Karte positiv beeinflusst.

7.3. RFID-Reader

Um eine Story auch einem Mitarbeiter zuweisen zu können, muss sich dieser einer Karte zuordnen können. Dieser Prozess soll möglichst einfach gehalten werden, um das Daily nicht unnötig zu verlängern. Dazu soll ein RFID-Reader mit einem eigenen Mikrocontroller am Board installiert werden, über den die einzelnen Teammitglieder ihren Teil des Dailys starten können.

7.3.1. RDM6300

Das Modul RDM6300 der Firma Seeedstudio besteht aus einem eigenen Board mit Ein-/Ausgängen. Dazu wird eine Spule, die ein elektromagnetisches Feld erzeugt, mit den Eingängen des Boards verbunden. Die Ausgänge werden mit dem entsprechenden Mikrocontroller verbunden. Diese Architektur hat den Nachteil, dass sie mehr Platz verbraucht und doppelt verkabelt werden muss. Jedoch lässt sich so die Spule bzw. das Board oder Kabel bei einem Defekt einfacher austauschen.

7.3.2. RC522

Der RFID-Reader RC522 ist von verschiedenen Händlern erhältlich und bietet die gleiche Funktionalität wie der RDM6300. Allerdings ist der Leser hier im Modul integriert. So muss nur einmal verkabelt werden. Außerdem nimmt er weniger Platz ein und bietet eine Verbindung über die SPI Schnittstelle an [25]. Insbesondere durch die Abmessungen eignet er sich hier besser und wird daher im Prototyp verwendet.

7.4. Veränderungen des Status mittels Sensorik

Um den Status einer Story zu erfassen, soll es möglich sein diesen an einer Karte zu verändern. Wichtig hierbei sind Minimierung möglicher Fehler, eine einfache Wartung und Portierbarkeit. Auch sollte die Lösung keine weitere Stromversorgung seitens der Karte benötigen.

7.4.1. Bestimmung der Position durch RFID-Tags

RFID-Reader können entsprechende Tags in sehr kurzer Zeit erkennen und ihre ID auslesen. Daher wäre es auch möglich das Board mit RFID-Folie zu bekleben und die Karten mit Reader auszustatten. Je nach dem um welche Spalte es sich handelt, besitzt diese eine andere ID. Sobald der Reader in der Karte eine neue ID erkennt, leitet er diese an den Server weiter, der dementsprechend den Status der Story verändert. Die Präparierung des Boards wäre ein aufwendiger Prozess. Auch müsste so jede Karte einen RFID-Leser besitzen, wodurch die Karte größer und unhandlicher wird. Außerdem steigt der Stromverbrauch der Karte.

7.4.2. Ausstatten der Karten mit Vibrationssensor

Vibrationssensoren messen Erschütterungen und geben diese als analogen Wert an einen Mikrocontroller weiter. Je höher die Bewegung, desto höher der entsprechende Wert. Ein solcher Sensor könnte in einer Karte angebracht werden. Wenn eine Karte z.B. in To Do hängt und der Vibrationssensor eine Bewegung vermerkt, kann die Karte diese an den Server weiterleiten. Der Server ändert den Status im JIRA dann auf In Work. Um die Interpretation von allgemeinen Erschütterungen nicht als Statusveränderung zu interpretieren, wird der Sensor erst aktiv, wenn sich ein Teammitglied per RFID am Daily anmeldet. Ein großes Problem ist hier das Zurückstellen von Stories und Karten. So kann ein Teammitglied eine Karte nicht in einen vorherigen Status überführen. Außerdem gibt es Dailies, in denen die Teammitglieder die Karte über die sie reden in die Hand nehmen, obwohl sie den Status nicht verändern. Aufgrund dieser Fehleranfälligkeit eignen sich Vibrationssensoren nicht.

7.4.3. Bestimmung der Position durch Infrarot

Infrarotsensoren messen eine Infrarotstrahlung und geben die entsprechende Frequenz zurück. Wenn man das Board mit Infrarotsendern (beispielsweise über jeder Spalte in Richtung Boden) auf verschiedenen Frequenzen ausstattet, könnte man diese Strahlung über die Karte auslesen. Allerdings ist dieses Verfahren ungenau, da die Sender die Strahlen nicht gezielt nach unten strahlen können, sondern eher streuen, wie z.B. bei Lampen. Daher ließe sich zwar mit einem Sensor eine Strahlung messen, jedoch könnte man diese nicht exakt einem Sensor zuordnen, wodurch diese Methode zu fehleranfällig ist.

7.4.4. Bestimmung der Position durch Bluetooth Beacons

Eine genauere Positionsmessung bieten Bluetooth Beacons an. Diese senden von ihrem Standort eine eindeutige Kennung, die sogenannte UID, ab. Diese kann ein Client (bspw. ein Smartphone) lesen und dadurch seinen Abstand zum Beacon ermitteln [26]. Mit mehreren Beacons ist so eine Positionsermittlung des Clients möglich.

Karten lassen sich mit eigenen Sensoren so auf dem Board lokalisieren. Das größte Problem stellt hier die Stromversorgung dar. So brauchen die einzelnen Beacons jeweils eine eigene Stromversorgung, aber auch die einzelnen Sensoren benötigen Strom von der Karte. Darüber hinaus ist das System schwieriger einzurichten, da die Beacons für jedes Board (je nach Größe und Anordnung der Spalten) neu konfiguriert werden müssen, insbesondere bei Boards, die nicht nur drei, sondern sechs oder mehr Spalten nebeneinander haben.

7.4.5. Schalter an der Karte zur manuellen Statusveränderung

Die technisch schlichteste Lösung ist ein zusätzlicher Taster, Knopf oder Schalter an einer Karte. Da dieser ein analoges Signal zurückgibt, kann er als Interrupt im Deep Sleep Mode verwendet werden, was den Stromverbrauch niedrig hält. Bei Knopfdruck kann der Status einer Karte "erhöht" werden, also von To Do auf In Work auf Done. Wenn man den Status erniedrigen will, kann man nach Done nochmals den Schalter betätigen, um wieder auf To Do zu kommen. Diese Umsetzung ist auf jedes Board übertragbar und benötigt keine besondere Installation oder Konfiguration. Auch lässt sich die Zuweisung damit identifizieren. Wenn nach einer Anmeldung im Daily der Knopf betätigt wird, wird der Status verändert und der angemeldete Mitarbeiter der Story zugewiesen. Wenn eine Story auf ein anderes Teammitglied übertragen werden soll, kann nach Anmeldung der Knopf betätigt werden. Wenn das entsprechende Mitglied dann noch nicht zugewiesen ist, wird es zugewiesen. Erst ein weiterer Knopfdruck ändert den Status.

Jedoch ist ein Schalter keine automatisierte Lösung, wodurch das Daily komplizierter und länger wird. Insbesondere das Verschieben in einen vorherigen Status muss somit mit dreimaligem Drücken kenntlich gemacht werden. Auch kann das System nicht erkennen wo sich eine Karte aktuell befindet, sondern es nur speichern. So sind alle Karte zu Sprintbeginn in To Do. Wenn nun ein Knopf betätigt wird, speichert das System dies, kann es aber nicht mit dem Board prüfen. Bei einer manuellen Statusänderung im JIRA kann eine Inkonsistenz zum physischen Board nur durch die gespeicherten Status auf dem Server geprüft werden.

Im Prototyp werden die Karten mit Schaltern ausgestattet. Das stellt jedoch keine dauerhafte Lösung dar. Wenn die Nutzer in der Evaluation Probleme mit der Benutzung bekommen, werden hier weitere Technologien und Ansätze in Erwägung gezogen. Die vierte Story "Status von Stories und Unteraufgaben verändern" lässt sich damit jedoch umsetzen.

7.5. Server als REST-API zur Kommunikation mit JIRA und Karte

Boards wie das ESP8266 besitzen einen verhältnismäßig geringen Speicher und können keine aufwendigen Prozesse ausführen. Aus diesem Grund soll ein möglichst großer Teil der Anwendungslogik auf einen Server verlagert werden, der sowohl mit den einzelnen Karten als auch mit JIRA kommuniziert. Weitere Systeme, die in Zukunft an das neue Board verknüpft werden, sollten ebenfalls mit diesem Server kommunizieren.

Es wird sich um einen REST-Server handeln, da sich der ESP8266 relativ einfach mit diesem verbinden kann [22]. Umgesetzt wird der Server im Javascript-Framework NodeJS mit Nutzung der Middleware Express. Über HTTP tauschen sich die Karte und das Terminal mit dem Server aus. Dazu stellt der Server verschiedene Endpunkte bereit, über die Daten gesendet und empfangen werden können. Die Daten werden im JSON-Format versendet, da sie sowohl vom Server, als auch von der Karte leicht zu verarbeiten sind [27].

Die Kommunikation mit dem digitalen Board findet über die "Jira Software Server REST API" statt [28]. Hier werden Informationen zu Stories und Unteraufgaben geladen, sowie Status und Bearbeiter von Stories und Unteraufgaben verändert und angepasst. Die Authentifizierung findet durch HTTP Basic mit SSL Verschlüsselung statt, wobei die Zugangsdaten auf dem Server gespeichert sind. Um Missbrauch durch Nutzung des Servers zu verhindern, muss sich dieser im selben Netz wie die Karte und das Terminal befinden und darf nicht von außen erreichbar sein. Ist dies doch der Fall, so müssen die Zugangsdaten in der Karte und dem Terminal gespeichert sein, sodass der Server diese an die JIRA API weiterleitet. Für den Prototyp reicht es aus die Daten serverseitig zu speichern und diesen nur in Betrieb zu nehmen, wenn das System getestet wird. Die Requests an das JIRA werden mit dem Node-Modul axios versendet [29].

7.6. Auswahl der JIRA API

Es gibt verschiedene Möglichkeiten mit einem JIRA programmatisch zu interagieren. So gibt es grundlegend zwei APIs, die genutzt werden können [28]. Die "Service Desk Server REST API" bietet die Möglichkeit Kundenanfragen zu verarbeiten und tangiert die Möglichkeit in JIRA Stories zu formulieren und abzurufen nicht und eignet sich daher nicht [30]. Darüber hinaus gibt es die "Server platform REST API", die die Möglichkeit gibt die JIRA Server-Anwendung anzusprechen, in der Stories und Issues gespeichert sind [31]. Daher wird diese API verwendet.

Um das digitale Board in Form von JIRA mit der Karte zu verknüpfen muss es möglich sein einer Story eine Karte zuzuweisen. Dazu soll eine Story eine neue Eingabekomponente, ein sogenanntes "Custom Field", erhalten [32]. In diesem Feld wird eine eindeutige ID eingetragen, die jede Karte vorher erhalten hat. So kann der Server die JIRA API nach einer Story und dem Custom Field anfragen und somit Karte und Story verbinden.

7.7. Stromversorgung der Karte

Die Versorgung der Karte erfolgt in der Entwicklungsphase stationär. Für ein im Alltag taugliches System muss hier eine Lösung gefunden werden. Hierbei sollte die Stromversorgung in der Karte integriert sein und mindestens die Länge eines Sprints halten. Die Nutzung von Akkus über die USB-Schnittstelle ist nicht möglich, da die verbrauchte Stromstärke des ESP zu gering ist, um z.B. im Deep Sleep versorgt zu werden. Im Rahmen der weiteren Entwicklung muss in der Bachelorarbeit hier eine Lösung seitens der Hardware gefunden werden.

8. Implementation des Prototypen

Im folgenden Kapitel wird die Umsetzung der User Stories in Form eines Prototypen erläutert. Hierbei liegt der Fokus auf der Anwendungslogik und den Anschluss von Sensoren an die jeweiligen Mikrocontrollern.

8.1. Umsetzung eines Anmeldeterminals zur Verwendung des Boards in Dailies

8.1.1. Benötigte Sensoren und angeschlossene Hardware

Wie in Kapitel 7.1.4 beschrieben wird die Anmeldung mit RFID-Tags und einem RC522 Leser an einem ESP8266 mit NodeMCU umgesetzt. Der RFID-Reader kommuniziert über das Serial Peripheral Interface, kurz SPI mit dem Mikrocontroller. Das SPI wird für eine synchron-serielle Verbindung zwischen einem Host (in dem Fall der Mikrocontroller) und einem Peripheriegerät (in dem Fall der Reader) verwendet [33]. Hierfür sind vier SPI-spezifische, sowie drei Mikrocontroller-spezifische Leitungen notwendig.

Mit der SCK-Verbindung ("Serial Clock") gibt der Host eine Zeit vor, um eine Synchronisation zu ermöglichen. Zum parallelen Austausch wird je eine Leitung pro Richtung benötigt. Dazu gibt es MISO ("Master in, Slave out") und MOSI ("Master out, Slave in"), wobei sich Master auf den Host und Slave auf das Peripheriegerät bezieht. Zusätzlich gibt es die SDA-Verbindung ("Serial Data"), die für eine I^2C Kommunikation notwendig ist. Die drei weiteren Anschlüsse 3.3V, GND ("Ground") und RST ("Reset") sind für Stromzufuhr, Erdung und Neustart notwendig, wie sie die meisten angeschlossenen Geräte benötigen. In der folgenden Abbildung ist die Verbindung von Reader und Board skizziert.

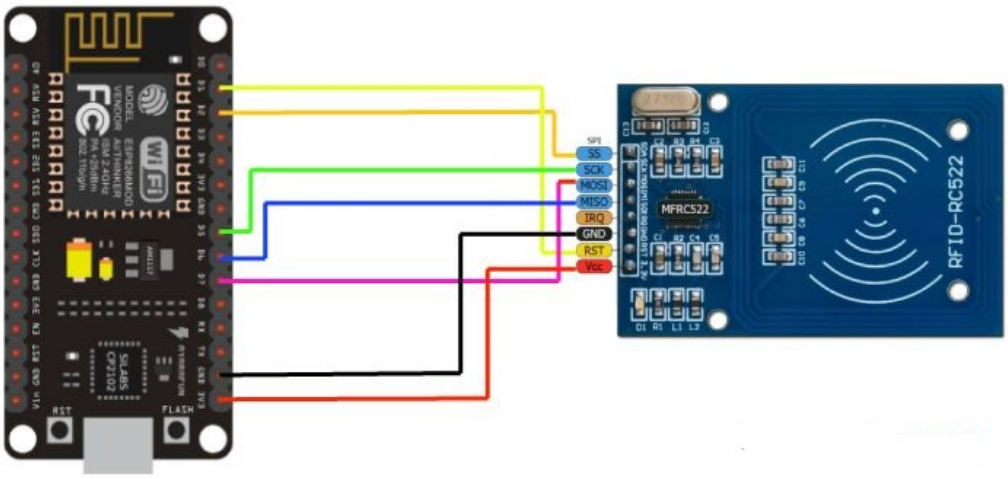

Abbildung 4: Verbindungsskizze zwischen dem RC522-Modul und dem ESP8266, Quelle: `https://www.instructables.com/id/MFRC522-RFID-Reader-Interfaced-With-NodeMCU/`

Außerhalb des RFID-Readers ist noch ein Button angeschlossen, der bei Betätigung

einen Stromkreis schließt und somit dem Mikrocontroller den Status "HIGH" und ansonsten den Status "LOW" signalisiert. Auch er ist mit GND und 3.3V, sowie einem GPIO-Pin (D2) verbunden, um das Signal weiterzuleiten.

8.1.2. Implementation der Anwendungslogik

Zur Einbindung des Readers wird sowohl die SPI-, sowie die MFRC522-Bibliothek benötigt. Diese bietet eine unkomplizierte Kommunikation mit dem Sensor an [34]. Dazu wird zu Beginn eine Instanz von MFRC522 erstellt und als Parameter der Pin für SDA, sowie für RST übergeben, da diese konfigurierbar sind. Wichtig ist hierbei nicht die digitalen Pinnummer des Boards, sondern die Nummer des GPIO-Pins zu übergeben, der sich vom digitalen Pin unterscheidet.

Alle weiteren Verbindung müssen nach Vorgabe der Bibliothek wie in Abbildung 4 beschrieben bleiben. Die Variablen "pressed" und "interruptPin", sowie die Methode "resetFunc" werden für die Nutzung des Buttons in Form einer Interrupt Service Routine benötigt, näheres dazu in der Setup-Methode. Um bei einer Anmeldung am Terminal die NUID des RFID-Tags zu speichern, wird ein Byte-Array benötigt, welches "tagUID" heißt.

```
#include <SPI.h>
#include <MFRC522.h>
#include <ESP8266WiFi.h>
#include <ESP8266HTTPClient.h>

#define RST_PIN  0          // GPIO0 = D3
#define SS_PIN   15         // GPIO15 = D8

const char* ssid = "                ";
const char* password =  "         ";
const byte interruptPin = 4; //GPIO4 = D2
boolean pressed = false;

MFRC522 rfid(SS_PIN, RST_PIN);

byte tagUID[4];
void ICACHE_RAM_ATTR logOff();
```

Abbildung 5: Terminalcode mit Instanzierung von globalen Variablen. Hinweis: SSID und Passwort des Drahtlosnetzwerkes wurden aus datenschutzrechtlichen Gründen entfernt.

Zu Beginn wird ein Interrupt auf dem GPIO-Pin des Buttons festgelegt. Ein Interrupt ist ein Programmteil, der bei einem bestimmten Event (hier: Drücken des Buttons) ausgelöst wird, den eigentlichen Programmteil unterbricht und stattdessen eine Ausweichmethode ausführt. In diesem Fall wird bei Druck des Buttons

die Methode logOff() aufgerufen, die lediglich die Variable pressed auf true setzt und im Cache verortet ist. Das ist notwendig, damit nicht nur der erste Knopfdruck registriert wird. Diese Interrupt Service Routine wird nur ausgeführt, wenn der Buttonstatus "RISING" ist, also der Wert steigend ist, was nur bei einem Knopfdruck der Fall ist. Die Variable pressed und somit die Information, dass der Button gedrückt wurde, wird in der loop-Methode verarbeitet.

Ebenfalls initiiert wird die serielle Verbindung zwischen dem Entwicklungsrechner und dem Mikrocontroller auf einer Rate von 115200 Baud. Sie ist für Logging und Fehlermeldungen nötig. So lässt sich am Rechner der USB-Port, an dem der Controller angeschlossen ist auf der Baud-Rate auslesen. Diese Verbindung wird mit Serial.begin() gestartet und mit Befehlen wie Serial.print("Text") angesprochen. Zur Weiterleitung der Terminalanmeldungen an den Server wird eine WLAN-Verbindung mit WiFi.begin(ssid, passwort) gestartet und auf eine erfolgreiche Verbindung (Status "WL_CONNECTED") gewartet. Während des Herstellens der Verbindung werden keine weiteren Programmteile ausgeführt. Nach erfolgreicher Verbindung werden das SPI und der RFID-Reader mit Methoden der jeweiligen Bibliotheken initiiert. Zum Schluss wird über die serielle Verbindung eine Meldung über das abgeschlossene Setup ausgegeben.

```
void setup() {
  attachInterrupt(digitalPinToInterrupt(interruptPin), logOff, RISING);
        Serial.begin(115200);
  WiFi.begin(ssid, password);

  while (WiFi.status() != WL_CONNECTED) {
    delay(1000);
    Serial.println("Waiting for WiFi Connection..");
  }

        SPI.begin();
        rfid.PCD_Init();
  delay(5);

  Serial.println("Setup complete");
}
```

Abbildung 6: Terminalcode der Setup-Methode

Die loop-Methode wird nach einmaliger Ausführung der setup-Methode in einer Schleife ausgeführt. Es gibt zwei Aktionen, auf die das Terminal reagieren muss: Das Betätigen des Buttons und das Anlegen eines RFID-Tags. Daher werden diese zwei Aktionen permanent abgefragt. Die Variable pressed beschreibt, ob der Button gedrückt wurde. Wenn dies der Fall ist, wird zusätzlich geprüft, ob das Terminal mit dem Internet verbunden ist. Wenn auch dies der Fall ist, wird der Server kontaktiert. Dazu wird ein PUT-Request an die Ressource "/logoff" versendet, näheres dazu in Kapitel 8.3.1. Der Server meldet den angemeldeten Benutzer ab und gibt dies als

Response zurück.

Die zweite zu prüfende Aktion ist das Anhalten eines Tags. Hierzu wird zuerst geprüft, ob eine Karte angehalten wird und ob eine Seriennummer auslesbar ist. Das Array tagUID speichert die UID des Tags in Bytes. Um keinen Tag doppelt hintereinander auszulesen, werden die Bytes des aktuellen Tags mit denen der Variable tagUID abgeglichen. Wenn sie sich nicht entsprechen, wird die UID des neuen Tags gespeichert und die Methode "sendUidToServer" aufgerufen.

```
void sendUidToServer() {
  unsigned long hex_num;
  hex_num =  tagUID[0] << 24;
  hex_num += tagUID[1] << 16;
  hex_num += tagUID[2] <<  8;
  hex_num += tagUID[3];

  if (WiFi.status() == WL_CONNECTED) {

      HTTPClient http;
      String hexString = String(hex_num);
      char url[128];
      char buf[32];
      hexString.toCharArray(buf, 32);
      strcpy(url, "http://jiracardserver.herokuapp.com/login/");
      strcat(url, buf);
      http.begin(url);

      int httpCode = http.PUT("");

      if (httpCode > 0) {
        Serial.println(http.getString());
      }
  }
}
```

Abbildung 7: Terminalcode der SendUidToServer-Methode

Diese Methode verarbeitet die UID und leitet sie an den Server weiter. Zu Beginn wird das Bytearray in einen Zahlenwert und dieser in einen String umgewandelt, um ihn einfacher zu versenden. Bei vorhandener Internetverbindung wird ein PUT-Request an die Ressource "/login/uid" gesendet, wobei "uid" die konvertierte UID des Tags ist. Die entsprechende Anmeldung eines Nutzers am Board wird mit der Response vom Server bestätigt. Der genaue Prozess von An- und Abmeldungen wird im Kapitel 8.3 erläutert.

8.2. Umsetzung einer digitalen Karte zur Darstellung von Stories und Unteraufgaben

8.2.1. Benötigte Sensoren und angeschlossene Hardware

Der ESP8266 mit NodeMCU für die Anzeige von Stories wird mit einem eInk-Display und einem Button verbunden. Die Verbindung mit dem Button verhält sich wie beim Terminal, auch was die Interrupt Service Routine betrifft. Das Display ist ebenfalls per SPI verbunden [35]. Hierbei steht seitens des Displays DIN für MOSI, CLK für SCK, CS für die SPI Chip Selection, DC für die SDA-Verbindung und BUSY für MISO. Darüber hinaus werden ebenfalls 3.3V, GND und RST verbunden, siehe hierzu auch das Terminal mit dem RFID-Reader. Während das Terminal permanent mit Strom versorgt werden kann, da es stationär ist, sollte die Karte mit einer Batterie betrieben werden. In der Entwicklungsphase des Protoypen ist dies noch nicht vorgesehen, wird aber im weiteren Verlauf während der Bachelorarbeit implementiert. In diesem Schritt wird auch ein Deep Sleep implementiert, der die Laufzeit der Karte trotz Batteriebetrieb verlängern soll.

8.2.2. Implementation der Anwendungslogik

Die Karte benötigt mehrere Bibliotheken. Zur Nutzung des Displays sind die AdafruitGFX- und die darauf aufbauende GxEPD2-Bibliothek (sowohl "_BW" für schwarz/weiß als auch "_3C" für Farbe) notwendig. Zur Verarbeitung der Daten vom Server wird die Bibliothek ArduinoJson verwendet, die den Umgang mit Daten im JSON-Format vereinfacht [27]. Ferner werden Bibliotheken für die Verwendung von WLAN und HTTP, sowie Schriftarten für das Display eingebunden.

Das Display muss zu Beginn initialisiert werden, hierzu werden einige frei wählbare Pins übergeben. Darüber hinaus werden globale Variablen für die WLAN-Verbindung, den Button und Interrupt-Pin wie beim Terminal festgelegt. Die Karte erhält zusätzlich eine eigene ID. Diese ist nötig, um vom Server erkannt und identifiziert zu werden. Da die Abkürzung der Story aus verschiedenen Methoden aufgerufen wird, wird sie global deklariert und beim Laden einer Story befüllt. Ebenfalls wird ein JSON-Objekt initiiert und 2048 Byte für dieses reserviert. Zusätzlich wird die Methode zum Zeichnen von Stories und zum Neustart des Boards deklariert.

```
#define ENABLE_GxEPD2_GFX 0

#include <GxEPD2_BW.h>
#include <GxEPD2_3C.h>
#include <Fonts/FreeSansBold18pt7b.h>
#include <Fonts/FreeSans18pt7b.h>
#include <Fonts/FreeSans12pt7b.h>
#include <ArduinoJson.h>
#include <ESP8266WiFi.h>
#include <ESP8266HTTPClient.h>;

#if defined (ESP8266)
GxEPD2_3C < GxEPD2_420c, GxEPD2_420c::HEIGHT / 2 > display
  (GxEPD2_420c(/*CS=D8*/ SS, /*DC=D3*/ 0, /*RST=D4*/ 2, /*BUSY=D2*/ 4));
#endif

const char* ssid = "                `";
const char* password = "        ";
const char* abbreviation = "";
const byte interruptPin = 12;
const char* cardID = "1234";
int pressed = false;
int Led = 2; //LED_BUILTIN = GPIO2
StaticJsonDocument<2048> doc;
void drawStory(const char &title, const char &abbr, const char &assignee);
void ICACHE_RAM_ATTR buttonClick();
```

Abbildung 8: Terminalcode mit Instanzierung von globalen Variablen. Hinweis: SSID und Passwort des Drahtlosnetzwerkes wurden aus datenschutzrechtlichen Gründen entfernt.

Auch für die Karte wird zu Beginn eine serielle Verbindung zum Logging hergestellt, der Interrupt-Pin definiert und eine WLAN-Verbindung aufgebaut. Bereits in der setup-Methode wird ein GET-Request an den Server gesendet. In diesem Request übergibt die Karte ihre ID an den Server und fragt an, ob im JIRA eine Story für die Karte hinterlegt ist. Der genaue Prozess seitens des Servers wird in Kapitel 8.3.1 erläutert. Die empfangenen Daten werden in der zuvor beschriebenen JSON-Variablen gespeichert. Ein Beispiel für eine solche JSON-Struktur findet sich in Anhang A.

```
void setup ()
{
  Serial.begin(115200);
  display.init(115200);
  display.hibernate();
  pinMode (interruptPin, INPUT_PULLUP);
  WiFi.begin(ssid, password);

  while (WiFi.status() != WL_CONNECTED) {
    delay(1000);
    Serial.println("Connecting to WiFi..");
  }

  attachInterrupt(digitalPinToInterrupt(interruptPin), buttonClick, RISING);
  HTTPClient http;

  http.begin("http://jiracardserver.herokuapp.com/card/issue/1234");

  int httpCode = http.GET();
  if (httpCode > 0) {
    deserializeJson(doc, http.getString());
    abbreviation = doc["abbreviation"];
  }
  Serial.println("Setup completed");
}
```

Abbildung 9: Kartencode der Setup-Methode

In der Loop-Methode befindet sich ausschließlich der Workflow bei Drücken des Buttons. Das bedeutet, dass sich die Karte nur bei Knopfdruck verändert. Dazu wird wie beim Terminal eine Variable abgefragt, die in der ISR für den Button verändert wird. Wenn der Button gedrückt wurde und eine WLAN-Verbindung besteht, wird der aktuelle Status des Dailies vom Server abgefragt. Das bedeutet, dass der Server prüft, ob aktuell eine Person am Terminal angemeldet ist oder nicht. Abhängig von der Antwort des Server führt die Karte verschiedene Aktionen durch.

```
void loop ()
{
  if (pressed == true) {
    if (WiFi.status() == WL_CONNECTED) {

      HTTPClient http;
      http.begin("http://jiracardserver.herokuapp.com/dailyStatus");
      int httpCode = http.GET();

      if (httpCode > 0) {

        String response = http.getString();
        if (response == "null") {
          char url[128];
          strcpy(url, "http://jiracardserver.herokuapp.com/card/issue/");
          strcat(url, cardID);
          http.begin(url);
          int httpResponseCode = http.GET();

          if (httpResponseCode > 0) {

            String response = http.getString();
            deserializeJson(doc, http.getString());
            const char* storyName = doc["name"];
            abbreviation = doc["abbreviation"];
            const char* storyAssignee = doc["assignee"];
            drawStory(storyName, abbreviation, storyAssignee);

          } else {

            Serial.print("Error on sending GET Request: ");
            Serial.println(httpResponseCode);
          }
```

Abbildung 10: Kartencode der Loop-Methode, Teil 1

Ist kein Teammitglied am Terminal angemeldet, liefert der Server "null" zurück. Das bedeutet für die Karte, dass aktuell kein Daily stattfindet und dementsprechend auch der Status der Story nicht verändert werden muss. Wird der Button außerhalb eines Dailies gedrückt, soll sich die Karte aktualisieren, die aktuelle Story vom Server anfragen und auf dem Display anzeigen. Hierzu fragt die Karte wie in der Setup-Methode, ob für ihre ID eine Story hinterlegt ist. Das vom Server empfangene JSON wird dann gespeichert und die Karte mit der Story in der Methode "drawStory" beschrieben, näheres dazu siehe unten.

```
        } else {
          Serial.println("Logged in user is " + response);
          if (abbreviation != "") {
            char url[128];
            strcpy(url, "http://jiracardserver.herokuapp.com/status/");
            strcat(url, abbreviation);
            http.begin(url);
            int httpResponseCode = http.PUT("");
            if (httpResponseCode > 0) {
              Serial.println("Status changed");
            }

          } else {
            Serial.println("Somebody is logged in, but no Story here to change Status");
          }

        }

      } else {

        Serial.print("Error on sending GET Request: ");
        Serial.println(httpCode);

      }

      http.end();

    } else {
      Serial.println("Error in WiFi connection");
    }

  }
  pressed = false;

}
```

Abbildung 11: Kartencode der Loop-Methode, Teil 2

Für den Fall, dass der Button im Daily betätigt wird, soll sich der Status der Karte verändern und die Story dem angemeldeten Teammitglied zugewiesen werden. Zu Beginn wird die Variable "abbreviation" geprüft, die das Kürzel der Story gespeichert. Ist kein Kürzel vorhanden, gibt es auch keine anzupassende Story. Ist jedoch ein Kürzel vorhanden, wird ein PUT-Request an den Ressource "/status/issueID" gesendet, wobei issueID das Kürzel der Story beinhaltet. Das Verändern des Status, sowie die Zuweisung im JIRA übernimmt der Server. Zuletzt wird die Karte aktualisiert, um die neuen Änderungen anzuzeigen.

```
void drawStory(const char* title, const char* abbr, const char* assignee)
{
  display.setFont(&FreeSans18pt7b);
  display.setTextColor(GxEPD_BLACK);
  int16_t tbx, tby; uint16_t tbw, tbh;
  display.getTextBounds(title, 0, 0, &tbx, &tby, &tbw, &tbh);
  uint16_t x = tbw;
  uint16_t y = tbh;
  display.setFullWindow();
  display.firstPage();
  do
  {
    display.fillScreen(GxEPD_WHITE);
    display.setCursor(x + 50, 0);
    display.println(abbr);
    display.setFont(&FreeSansBold18pt7b);
    display.println(title);
    display.setCursor(x + 50, display.height() - 50);
    display.setFont(&FreeSans12pt7b);
    display.print(assignee);
  }
  while (display.nextPage());
  display.powerOff();
  Serial.println("Display write completed");
}
```

Abbildung 12: Kartencode der drawStory-Methode

Um das Display mit neuen Informationen zu beschreiben gibt es die Methode "drawStory". Diese erhält den Titel, die Abkürzung und den Bearbeiter der Story und zeigt diese auf dem Display an. Dazu werden verschiedene Methoden der GxEPD2-Bibliothek verwendet, die den Umgang und die Nutzung des Displays vereinfachen. Zu Beginn wird das Display auf einer bestimmten Baud-Rate initiiert, sowie Schriftart, Schriftfarbe und Grenzen des Textes festgelegt. Das Display wird zunächst weiß gefüllt, um die bisherige Anzeige zu überschreiben. Daraufhin wird der Cursor 50 Pixel vom linken Rand entfernt platziert und die Abkürzung ausgegeben. In der nächsten Zeile folgt der Titel der Story. Für den Bearbeiter wird der Cursor am rechten unteren Rand platziert, sodass der Bearbeiter separat hinzugefügt. Hierzu wird die Schriftgröße von 18 Punkt auf 12 Punkt verändert. Die Angabe der Schriftgröße erfolgt als eigene Schriftart mit der Größe angehangen. Zuletzt wird das Display ausgeschaltet und in den Deep Sleep versetzt, um Strom bis zur nächsten Aktualisierung zu speichern.

8.3. Umsetzung eines NodeJS-Servers als Verbindung zwischen Board und JIRA

8.3.1. Erstellung von REST-URIs als Schnittstelle für digitale Karte und Anmeldeterminal

Zur Kommunikation mit Karte und Terminal stellt der Server mehrere Pfade und REST-Ressourcen zur Verfügung, die über HTTP-Requests angesprochen werden können. Die Response-Daten vom Server werden stets als JSON (content-type: application/json) zur Verfügung gestellt. Die verschiedenen JSON-Strukturen werden in Anhang A aufgelistet.

Die Ressource "/card/issue/:cardId" nimmt in der Variable "cardId" die ID einer Karte entgegen und liefert als Response die im JIRA zugewiesene Story zurück. Zuerst wird vom Server ein GET-Request an die JIRA API versendet, der einen Befehl in der "JIRA Query Language" (kurz JQL) enthält, der das gesamte JIRA nach Stories durchsucht, die das Feld "card_id" enthalten und der ID der Karte entsprechen [36].

```
app.get('/card/issue/:cardId', function (req, res) {
    axios({
        method: 'get',
        url: 'https://jira.        .de/rest/api/2/search?jql=card_id~' + req.params.cardId,
        auth: {
            username:  .       ',
            password: '            '
        }
    }).then(function (response) {
        issueId = response.data['issues'][0]['key'];
        axios({
            method: 'get',
            url: 'https://jira.        .de/rest/api/2/issue/' + issueId,
            auth: {
                username: '       ',
                password: '            '
            }
        }).then(function (response) {
            res.send(filterData(response.data["fields"], issueId));
        });
    }).catch(function (error) {
        console.log(error);
    });
});
```

Abbildung 13: Servercode der "card/issue"-Ressource, Teil 1. Hinweis: URL des JIRA, Nutzername und Passwort wurden aus datenschutzrechtlichen Gründen entfernt.

Da das Custom Field card_id ein Freitextfeld ist, ist es möglich bei mehreren Stories

die gleiche ID zu hinterlegen. Bei einer Suche mittels JQL werden alle Ergebnisse aufgelistet. Diese werden einer Relevanzsortierung durch das JIRA unterzogen bevor sie versendet werden. Das sorgt dafür, dass die Karte (meist) die Story enthält, die am aktuellsten ist. Für die weitere Sortierung wird hier eine zusätzliche Variable in der JIRA Query nötig sein, um hier auch die aktuellste Story zu garantieren. Generell empfiehlt es sich die card_id nach der Nutzung der Karte wieder zu entfernen.

```
app.get('/issue/:issueId', function (req, res) {
    axios({
        method: 'get',
        url: 'https://jira.        .de/rest/api/2/issue/' + req.params.issueId,
        auth: {
            username: '       ',
            password: '             '
        }
    }).then(function (response) {
        res.send(filterData(response.data["fields"], req.params.issueId));
    }).catch(function (error) {
        console.log(error);
    });
});
```

Abbildung 14: Servercode der "/issue"-Ressource. Hinweis: URL des JIRA, Nutzername und Passwort wurden aus datenschutzrechtlichen Gründen entfernt.

In den Suchergebnissen erhält man entsprechende Stories und einige weitere Informationen. Allerdings fehlen in den Suchergebnissen Informationen zu Bearbeiter und Status. Daher wird das Kürzel der ersten Story der Suchergebnisse in der Variable "issueId" gespeichert. Daraufhin wird ein weiterer Request an die Ressource "/issue/:issueId" abgesendet mit der zuvor ermittelten ID.

```
function filterData(data, abbreviation) {
    let filteredData = {};
    if (data["issuetype"]["name"].toLowerCase() === "story") {
        filteredData.name = data["summary"];
        filteredData.abbreviation = abbreviation;
        filteredData.assignee = data["assignee"]["displayName"];
        filteredData.status = data["status"]["name"];
        filteredData.issuetype = "Story";
    } else if (data["issuetype"]["name"].toLowerCase() === "unteraufgabe") {
        filteredData.name = data["summary"];
        filteredData.abbreviation = abbreviation;
        filteredData.assignee = data["assignee"]["displayName"];
        filteredData.status = data["status"]["name"];
        filteredData.issuetype = "Unteraufgabe";
    } else {
        console.error("Issuetype = " + data["issuetype"]["name"]);
        filteredData.name = "Error!";
    }
    updateLocalIssues(filteredData);
    return filteredData;
}
```

Abbildung 15: Servercode der ”filterData”-Methode

Da die Response jedoch sehr viele für die Karte irrelevante Informationen enthält, werden diese in der Funktion ”filterData” gekürzt und lokal auf dem Server gespeichert. Das ist notwendig, um bei Knopfdruck und Statusänderung den aktuellen Status ohne API-Call zu ermitteln und somit Wartezeit zu sparen. Dazu wird in der Funktion ”updateLocalIssues” lediglich das JSON der aktuellen Stories vom Server geladen und um die neu ermittelte Story erweitert, falls sie nicht schon vorhanden ist. Zuletzt werden die gefilterten Daten an die Karte gesendet, Struktur siehe Anhang A.1. Die Ressource /issue/:issueId führt lediglich den zweiten Schritt aus und ermittelt Informationen zum Kürzel einer Story in der Variable issueId.

```
app.put('/status/:issueId', function (req, res) {
    if (activeUser !== "null") {
        let transitionId = increaseStatus(req.params.issueId);
        axios({
            method: 'post',
            url: 'https://jira.        .de/rest/api/2/issue/'
                + req.params.issueId + '/transitions?expand=transitions.fields',
            auth: {
                username: '       ',
                password: '            '
            },
            data: {
                transition: {
                    "id": transitionId
                }
            }
        }).then(function (response) {
        }).catch(function (error) {
            console.error(error['response']['data']['errorMessages']);
```

Abbildung 16: Servercode der "/status"-Ressource, Teil 1. Hinweis: URL des JIRA, Nutzername und Passwort wurden aus datenschutzrechtlichen Gründen entfernt.

Für das Ändern eines Status gibt es die PUT-Ressource "/status/:issueId", wobei die "issueId" das Kürzel der Story beschreibt. Zu Beginn wird ermittelt, ob ein Teammitglied zum Daily angemeldet ist. Dies ist in der Variable "activeUser" festgehalten. Ist niemand zum Daily angemeldet, wird die Variable auf "null" gesetzt. Der Status einer Story soll nur verändert werden können, wenn ein Daily stattfindet und somit jemand angemeldet ist. In diesem Fall wird zuerst anhand des Kürzels der aktuelle Status ermittelt, indem die lokal gespeicherten Stories nach der issueId durchsucht werden und der Status gesucht wird. In JIRA geschieht dies durch sogenannte Transitions, die in jedem JIRA-Projekt selbst du den Administrator definiert werden können A.2. Jede diese Transitions hat eine eindeutige ID. Auf dem Server ist die Zuordnung von Status zu TransitionId in Form eines JSON gespeichert, siehe Anhang A.2. Der Server findet die TransitionId des nächsten Status und speichert diese in der Variable "transitionId".

```
}).then(function () {
    axios({
        method: 'put',
        url: 'https://jira.        .de/rest/api/2/issue/' + req.params.issueId,
        auth: {
            username: '       ',
            password: '            '
        },
        data: {
            "fields": {
                "assignee": {
                    "name": activeUser
                }
            }
        }
    }).then(function (response) {
    }).catch(function (error) {
        console.error(error['response']['data']['errorMessages']);
    });
    let responseString = "Forwarding status of "
        + req.params.issueId + " assigned to " + activeUser;
    res.send(responseString);
});
```

Abbildung 17: Servercode der "/status"-Ressource, Teil 2. Hinweis: URL des JIRA, Nutzername und Passwort wurden aus datenschutzrechtlichen Gründen entfernt.

Mit dieser ID wird die POST-Ressource "/issue/:issueId/transitions" der JIRA API aufgerufen und die neue Transition übergeben. Daraufhin wird (unabhängig vom Erfolg des vorherigen Requests) der neue Bearbeiter der Story über die PUT-Ressource "issue/:issueId" und dem übergebenen Feld "assignee" geändert. Falls bereits ein Bearbeiter für diese Story vorhanden ist, wird dieser überschrieben. Zuletzt wird als Response eine Erfolgsmeldung mit dem Kürzel der Story und dem Bearbeiter an die Karte gesendet.

```
app.put('/login/:userId', async function (req, res) {
    activeUser = getUserById(req.params.userId);
    let responseString = "User " + activeUser + " logged in successfully";
    res.send(responseString);
    await startDaily();
});

app.put('/logoff', function (req, res) {
    let responseString = "User " + activeUser + " logged off successfully";
    activeUser = "null";
    res.send(responseString);
});

app.get('/dailyStatus', function (req, res) {
    res.send(activeUser);
});
```

Abbildung 18: Servercode der "/login", "/logoff" und "/dailyStatus"-Ressourcen

Für die Karte gibt es drei verschiedene Ressourcen. Zum Anmelden eines Teammitgliedes wird die PUT-Ressource "/login/:userId" verwendet. Die "userId" entspricht hierbei der UID des entsprechenden RFID-Tags. Die Zuordnung von userId und Benutzernamen im JIRA ist serverseitig in einem JSON gespeichert, siehe dazu Anhang A.3. Die Funktion "getUserById" nutzt diese Zuordnung und gibt den JIRA-Benutzernamen zurück, welcher dann als activeUser gesetzt wird. Eine Erfolgsmeldung mit dem angemeldeten Teammitglied wird zurück an das Terminal gesendet. Zuletzt wird die Funktion "startDaily" aufgerufen, die als asynchronen Prozess einen Timer von 10 Minuten verstreichen lässt, bevor sie den activeUser zurück auf "null" setzt und das Teammitglied somit automatisch nach 10 Minuten abmeldet.

```
async function startDaily() {
    return new Promise(function () {
        setTimeout(function () {
            activeUser = "null";
        }, 600000);
    });
}
```

Abbildung 19: Servercode der "startDaily"-Funktion

Durch Knopfdruck am Terminal kann ein Teammitglied auch manuell abgemeldet werden. Hierzu wird die PUT-Ressource "/logoff" zur Verfügung gestellt. Sie setzt den activeUser direkt auf "null" und gibt eine Erfolgsmeldung zurück. Falls noch ein Timer läuft, wird dieser weiter ausgeführt und setzt nach Ablauf ebenfalls "null". Falls sich nach einem Abmelden erneut ein Teammitglied anmeldet, wird der Timer neu gestartet, sodass das Teammitglied trotzdem zehn Minuten angemeldet bleibt.

Um nun zu ermitteln, ob jemand angemeldet ist, gibt es die GET-Ressource "/dailyStatus". Diese gibt als Response lediglich den activeUser zurück. Dies ist für die Karte bei Knopfdruck relevant, aber darüber hinaus auch zum Testen des Servers hilfreich. Die weiteren Helfermethoden, globalen Variablen und Instanzierungen des Server wurden in diesem Kapitel nicht näher erläutert, sind aber auf den Abbildungen 23, 24, 25 und 26 in Anhang B zu finden.

8.3.2. Tabellarische Auflistung der REST-URIs

Die zuvor erläuterten Ressourcen werden in der folgenden Tabelle mit einer kurzen Beschreibung und ihrer REST-Methode aufgelistet.

Ressource	**Methode**	**Beschreibung**
/card/issue/:cardId	GET	Ermittlung einer Story durch cardId
/issue/:issueId	GET	Ermittlung einer Story anhand ihrer Abkürzung
/status/:issueId	PUT	Verändert den Status und Bearbeiter einer Story
/login/:userId	PUT	Meldet ein Teammitglied zum Daily am Board an
/logoff	PUT	Meldet ein angemeldetes Teammitglied wieder ab
/dailyStatus	GET	Ermittelt angemeldete Teammitglieder

8.3.3. Schnittstellen zur JIRA Server platform REST-API

In der folgenden Tabelle werden die Ressourcen der JIRA Server platform API aufgelistet, die vom Server angefragt und verwendet werden.

Ressource	**Methode**	**Beschreibung**
/search?jql=card_id∼:cardId	GET	Suche von Stories mit einer KartenID
/issue/:issueId	GET	Ermittlung einer Story
/issue/:issueId/transitions	POST	Verändert den Status einer Story
/issue/:issueId	PUT	Verändert den Bearbeiter einer Story

9. Evaluation anhand der User Stories

Im folgenden Kapitel wird das System mit Hilfe der in Kapitel 5 formulierten User Stories evaluiert. Hierzu werden insbesondere die Akzeptanzkriterien mit dem Verhalten des Systems verglichen und gegebenenfalls Maßnahmen abgeleitet, um es weiter zu verbessern.

9.1. Story einer Karte zuweisen

Der Karte wird wie gefordert eine ID zugewiesen. Dies passiert aktuell jedoch nicht automatisch. Das bedeutet, dass diese ID im Quellcode der Karte gespeichert ist und somit für jede Karte händisch eingetragen werden muss. Dies ist nötig, da die Karte bei einem Neustart (z.B. nach Unterbrechung der Stromzufuhr) ihre zugewiesenen Variablen verliert. Wenn die ID der Karte dynamisch auf dem Server generiert wird, muss es eine Möglichkeit geben diese auch serverseitig der Karte zuzuordnen. Denn wenn sich die Karte neu startet und ihre zuvor generierte ID beim Server anfragt, dann kann dieser die Karte nicht identifizieren und somit ihre ID nicht ausfindig machen. Hier gibt es Verbesserungspotential. Für den Prototypen erhält jede Karte ihre ID durch den Quellcode.

Die Möglichkeit eine Karte mit einer Story, bzw. Unteraufgabe zu verknüpfen besteht. Im in Kapitel 7.6 erläuterten Custom Field lässt sich eine ID eintragen. Außerdem kann die Karte mit der Ressource "/card/issue/:cardId" anhand ihrer ID erfragen, ob eine Story für sie hinterlegt ist. Bei jeder dieser Abfragen hinterlegt der Server in einem JSON die Zuordnung von Karten-ID und Informationen zur Story, siehe auch Anhang A.1.

Somit wurde die Story "Story einer Karte zuweisen" erfüllt. Alle Akzeptanzkriterien wurden implementiert, jedoch lässt sich die Umsetzung noch verbessern, sodass das System einfacher auf viele Karten portierbar ist.

9.2. Stories und Unteraufgaben auf einer Karte anzeigen

Wie in Anhang E zu sehen kann die Karte Stories anzeigen. Die Abkürzung wird über dem Titel der Story angezeigt, zusätzlich wird der Bearbeiter am unteren Rand abgebildet. Diese Informationen werden, wie im vorherigen Abschnitt bereits erwähnt, aus dem JIRA ermittelt und durch das E-Ink Display auch permanent angezeigt.

Verfehlt wurde das Akzeptanzkriterium die Karte regelmäßig zu aktualisieren. Um die neusten Informationen für die Karte zu laden, muss der Knopf an der Karte betätigt werden. Andernfalls aktualisiert sich die Karte nicht, anders als in der Story gefordert. Der einfache Weg dies umzusetzen wäre ein Zeitintervall, nach dem die Karte jedes Mal den Server anfragt und gegebenenfalls das Display aktualisiert. Allerdings gibt es (der Karte nicht bekannte) Zeiträume an denen eine Anfrage an den Server nicht nötig ist, wie z.B. an gesetzlichen Feiertagen oder nachts. Eine andere Möglichkeit wäre die Implementation einer Nachrichtenwarteschlange. Hierbei versendet der Server Nachrichten zu bestimmten Themen, die die einzelnen Karten verfolgen können, nach dem publish/subscribe-Prinzip.

Im Groben wurde die Story erfüllt und umgesetzt, wobei es Abstriche bei der Aktualisierung gibt. Eine Weiterentwicklung in diese Richtung ist für das Praxisprojekt nicht mehr vorgesehen, allerdings für die Bachelorarbeit denkbar. Für den Prototypen reicht die manuelle Aktualisierung aus.

9.3. Stories und Unteraufgaben einem Teammitglied zuweisen

Die Zuweisung von Teammitgliedern wurde durch das Terminal umgesetzt. Dadurch kann sich jedes Teammitglied anmelden und sich Stories zuweisen, jedoch nur wenn auch der Status der Story verändert wird. Das erfüllt zwar das Akzeptanzkriterium, schränkt das Team allerdings in ein, da zur Veränderung des Bearbeiters stets eine Statusänderung der Story erfolgen muss. Wenn ein Teammitglied z.B. durch Krankheit ausfällt und ein anderes Teammitglied seine Aufgaben übernimmt, muss der Status verändert werden, zur Not durch dreimaligen Knopfdruck, um den ursprünglichen Status wiederherzustellen.

Die Authentifizierung ist durch die RFID-Tags und die Zuordnung in Anhang A.3 umgesetzt. Ein angemeldetes Teammitglied wird bei Statusveränderung der Story an den Server weitergeleitet. Dieser weist die Story mittels der Ressource "/issue/:issueId" dem richtigen Teammitglied zu. Das Anzeigen des neuen Bearbeiters fehlt noch. Um dies umzusetzen muss die Karte automatisch nach der Anpassung aktualisiert werden.

Die Funktion der Zuweisung wurde umgesetzt und kann verwendet werden. Allerdings gestaltet sich diese umständlich, wenn der Status der Story nicht verändert werden muss. Zusätzlich fehlt eine automatische Aktualisierung der Karte nach Veränderung des Bearbeiters.

9.4. Status von Stories und Unteraufgaben verändern

Die Statusveränderung ist durch Knopfdruck und anschließender Weiterleitung an den Server durch die Karte möglich. Auch Fehler sind damit abgedeckt, da ein Teammitglied an einen vorherigen Status zurückkehren kann durch mehrmaliges Betätigen. Dieser Ablauf lässt sich jedoch noch verbessern. So könnte der Knopfdruck durch Sensorik ersetzt werden, bei der die Karte automatisch erkennt, in welchem Status sie sich befindet. Damit würde dem Team der Umgang mit dem Board erleichtert werden. Die genaue Technologie muss abgewägt werden, um einerseits den Stromverbrauch der Karte gering zu halten und andererseits die Anpassungen am Board klein zu halten, um das System einfach auf verschiedenen physischen Boards zu installieren und konfigurieren. Ebenfalls fehlt es hier an einem Feedback bei Knopfdruck, sodass den Teammitgliedern bewusst ist, dass die Karte den Input erhalten und verarbeitet hat. Hier würde ein Signal beispielsweise per LED oder Lautsprecher Teammitglieder unterstützen.

Generell wurde diese Story mit einfachen Mitteln erfüllt, jedoch gibt es hier ein großes Potenzial zur Verbesserung in der Bachelorarbeit.

10. Fazit

In diesem Kapitel wird diese Dokumentation anhand der Vorgehensweise zusammengefasst. Daraufhin wird die in Kapitel 2.5 erstellte Forschungsfrage beantwortet. Zum Abschluss wird das weitere Vorgehen in der Bachelorarbeit kurz beschrieben.

10.1. Zusammenfassung der Arbeit

Zu Beginn dieser Arbeit wurde die Problemstellung der Synchronisierung der physischen und digitalen Boards erläutert und einzelne Lösungsansätze diskutiert, wobei eine bidirektionale Synchronisation beider Boards gewählt wurde. Nach Festlegung der Forschungsfrage "Wie lassen sich analoge und digitale Scrum-Boards automatisch synchronisieren?" wurden Ziele definiert, um die Problemstellung mit dem ausgesuchten Grundgedanken zu lösen. Daraufhin wurden bisher verfügbare Produkte auf dem Markt hinsichtlich dieser Ziele analysiert und abgewägt. Aus den Schwächen jener entstanden Anforderungen im "User Story"-Format. Zur Erfüllung dieser Stories wurde die notwendige Anwendungslogik, sowie dadurch entstehende Systemarchitektur erläutert und skizziert. Für die einzelnen Komponenten des zuvor beschriebenen Systems wurden verschiedene Technologien in Erwägung gezogen und sich schlussendlich für passende Lösungen zur Umsetzung der User Stories entschieden. Nachdem der Prototyp des Systems implementiert wurde, wurde dies anhand des Quellcodes erläutert und erklärt. Abschließend wurde das System mit Hilfe der User Stories evaluiert. Dabei ergab sich, dass das System zum größten Teil alle Stories und Akzeptanzkriterien erfüllt hat, wobei es an einigen Stellen noch Möglichkeiten zur Verbesserung des Systems gibt. Insgesamt wurden die erarbeiteten Ziele jedoch erreicht.

10.2. Beantwortung der Forschungsfrage

"Wie lassen sich analoge und digitale Scrum-Boards automatisch synchronisieren?"

In Kapitel 4 wurden verschiedene Produkte auf dem Markt und Lösungsansätze beleuchtet. Die dort vorgestellten Alternativen sind in der Lage die Synchronisation von analogen und digitalen Boards zu unterstützen. Die automatische Synchronisation kann mit dem in Kapitel 8 umgesetzten System am besten erzielt werden, da es Alleinstellungsmerkmale besitzt, die das Problem direkt adressieren. Der umgesetzte Prototyp ist in seiner Funktionsweise und der Nutzung im Alltag noch beschränkt, zeigt jedoch die Möglichkeiten auf, die eine vollständige automatisierte Synchronisation erreichen können und beantwortet somit die Forschungsfrage.

10.3. Ausblick zur Bachelorarbeit

Um aus dem Prototypen ein im Alltag nutzbares System zu entwickeln wird in der Bachelorarbeit eine leichtgewichtige Evaluation in Form von Befragungen innerhalb des Unternehmens kernpunkt durchgeführt. Das Ergebnis dieser Evaluation wird in Anforderungen formuliert, welche mit der Weiterentwicklung des Systems erfüllt werden sollen. Nach der Umsetzung wird das System anhand der Anforderungen bewertet und ein Fazit für die Nutzbarkeit des Systems gezogen.

A. JSON-Strukturen

A.1. Informationen zur Story


```
[
  {
    "name":"PoCs in Latex formatieren",
    "abbreviation":"PPBA-7",
    "assignee":"Alexander Strutz",
    "status":"In Arbeit",
    "issuetype":"Story"
  },
  {
    "name":"Technologieentscheidung",
    "abbreviation":"PPBA-9",
    "assignee":"Alexander Strutz",
    "status":"Aufgaben",
    "issuetype":"Story"
  }
]
```


Abbildung 20: JSON mit weiteren Informationen einer Beispielstory

A.2. Zuordnung TransistionsId zu Status


```
{
  "projects": [
    {
      "abbreviation": "PPBA",
      "Aufgaben": 11,
      "Wird Ausgeführt": 21,
      "Fertig": 31
    }
  ]
}
```


Abbildung 21: JSON zur Zuordnung von JIRA-TransitionId zu Status in einem Beispielprojekt

A.3. Zuordnung UID zu Username


```
{
  "user": [
    {
      "id": "1053237465",
      "name": "astrutz"
    },
    {
      "id": "2495680030",
      "name": "sschneider"
    }
  ]
}
```


Abbildung 22: JSON zur Zuordnung von UID der RFID-Tags zu JIRA-Nutzernamen

B. Ergänzungen zur Server-Implementation

```
const express = require('express');
const axios = require('axios');
const fs = require('fs');
var myParser = require("body-parser");
let activeUser = "null";
let app = express();
let issueId = '';
let PORT = process.env.PORT || 3000;
app.use(myParser.json({extended: true}));
```

Abbildung 23: Servercode der globalen Variablen und Instanzierungen

```
app.listen(PORT, function () {
    console.log('Server listening on port ' + PORT + '!');
});
```

Abbildung 24: Servercode der Initialisierung

```
function updateLocalIssues(issue) {
    let found = false;
    let savedIssues = JSON.parse(getSavedIssues());
    for (let i in savedIssues) {
        if (savedIssues[i]["abbreviation"] === issue ["abbreviation"]) {
            savedIssues[i] = issue;
            found = true;
        }
    }
    if (!found) {
        savedIssues.push(issue);
    }
    setSavedIssues(savedIssues);
}
```

Abbildung 25: Servercode der Aktualisierung der lokal gespeicherten Stories

```
function getUserById(userId) {
    let userMap = JSON.parse(fs.readFileSync("rfidUser.json", 'UTF8'));
    for (let i in userMap['user']) {
        if (userMap['user'][i]['id'] === userId) {
            return userMap['user'][i]['name'];
        }
    }
}

function getSavedIssues() {
    return fs.readFileSync("savedIssues.json", 'UTF8');
}

function setSavedIssues(issues) {
    fs.writeFileSync('savedIssues.json', JSON.stringify(issues), 'UTF-8');
}

async function startDaily() {
    return new Promise(function () {
        setTimeout(function () {
            activeUser = "null";
        }, 600000);
    });
}
```

Abbildung 26: Servercode der weiteren Hilfsmethoden

C. Physisches Board

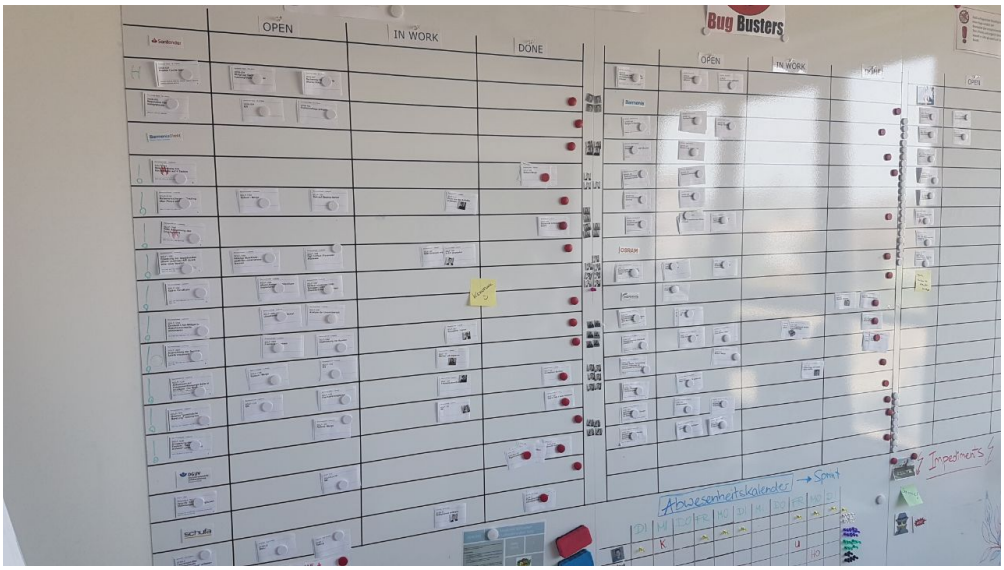

Abbildung 27: Beispiel eines physischen Boards im Unternehmen kernpunkt

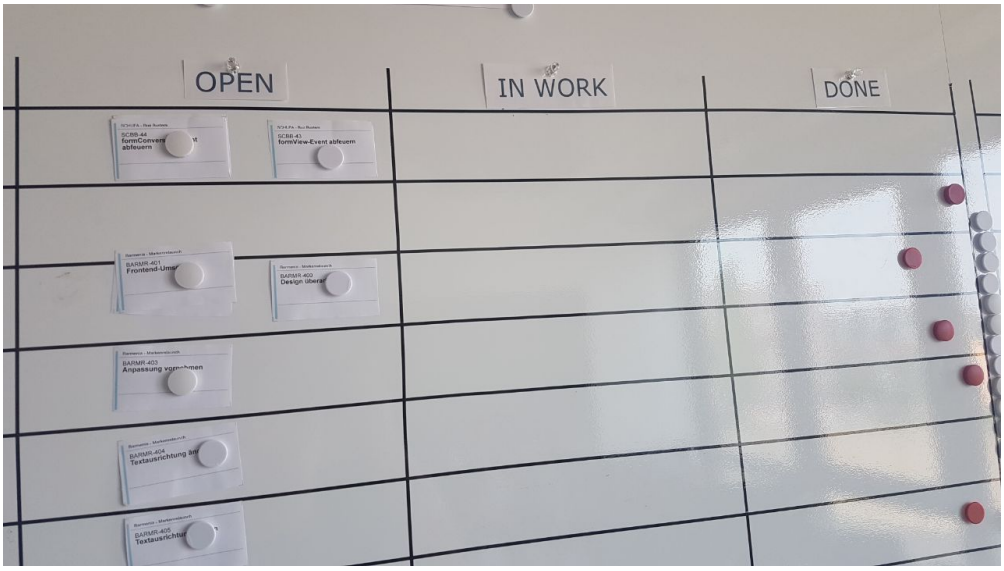

Abbildung 28: Spalten des physischen Boards

D. Digitales Board

Abbildung 29: Darstellung einer Story in JIRA

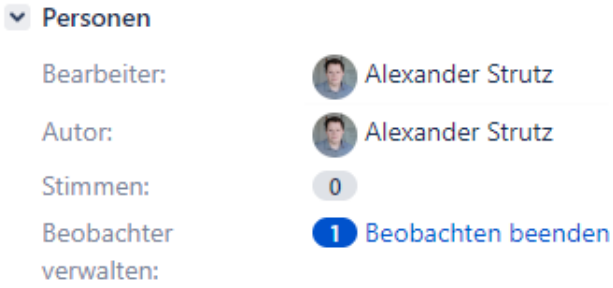

Abbildung 30: Darstellung eines Bearbeiters in JIRA

Abbildung 31: Darstellung von Unteraufgaben in JIRA

E. Prototyp

E.1. Karte

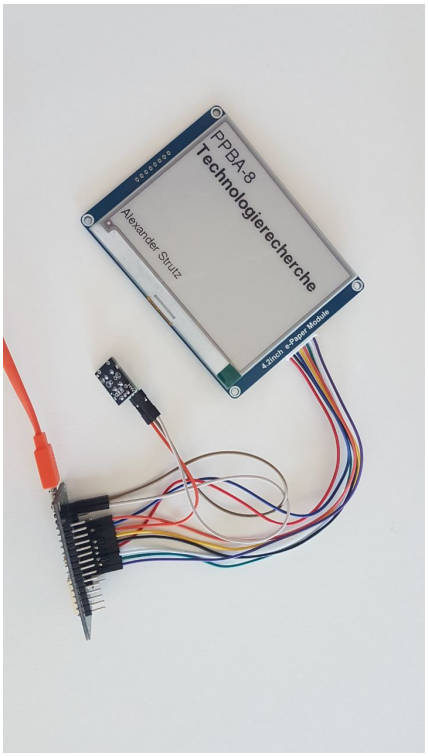

Abbildung 32: Prototyp zur Darstellung von Stories und Unteraufgaben

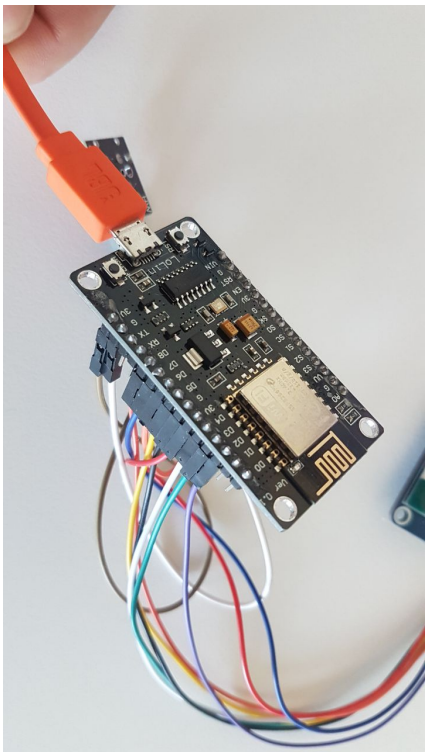

Abbildung 33: Nahaufnahme mit Blick auf die verwendeten GPIO-Pins

E.2. Terminal

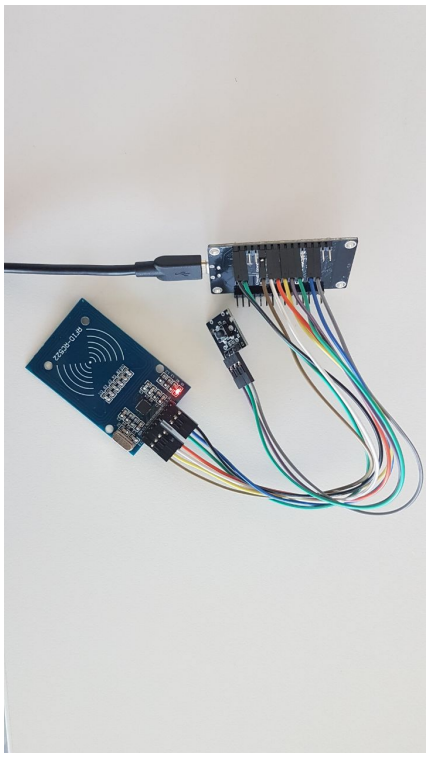

Abbildung 34: Prototyp zur Anmeldung von Teammitgliedern

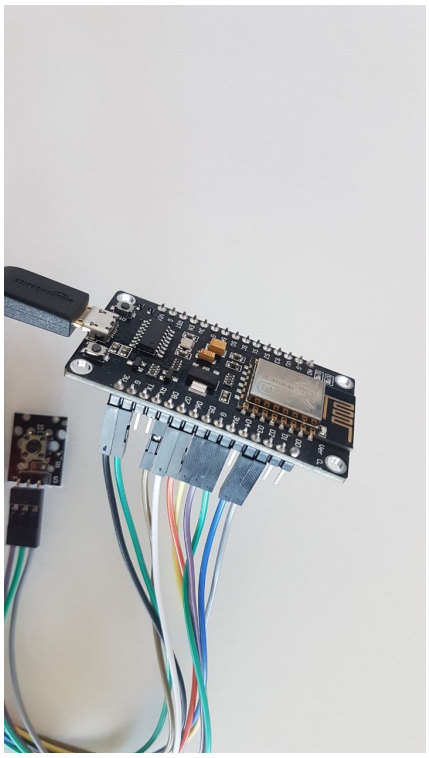

Abbildung 35: Nahaufnahme mit Blick auf die verwendeten GPIO-Pins

Literatur

[1] Hochschule Koblenz. Status quo agile 2016/2017. https://www.process-and-project.net/studien/status-quo-agile. Zugriff: 21. Februar 2019.

[2] Agile Alliance. Agile Manifesto. http://agilemanifesto.org/iso/de/manifesto.html. Zugriff: 21. Februar 2019.

[3] Ken Schwaber and Jeff Sutherland. The Scrum Guide. https://www.scrumguides.org/docs/scrumguide/v2017/2017-Scrum-Guide-US.pdf. Zugriff: 21. Februar 2019.

[4] Pam A. Mueller and Daniel M. Oppenheimer. The pen is mightier than the keyboard: Advantages of longhand over laptop note taking. In *Psychological Science 25(6)*, pages 1159–1168, London, 2014. SAGE.

[5] Teresa Amabile and Steven Kramer. *The Progress Principle - Using Small Wins to Ignite Joy, Engagement, and Creativity at Work*. Harvard Business Press, Boston, Massachusetts, 2011.

[6] Christian Katsma, Chintan Amrit, Jos van Hillegersberg, and Klaus Sikkel. Can agile software tools bring the benefits of a task board to globally distributed teams? In *International Workshop on Global Sourcing of Information Technology and Business Processes*, pages 167–174, Utrecht, 2015. School of Management and Governance, University of Twente.

[7] Thomas Perry. Drifting toward invisibility: The transition to the electronic task board. In *AGILE Conference*, pages 496–500, Toronto, 2008. IEEE Computer Society.

[8] Macrotrends LLC. Atlassian Revenue 2014-2018. https://www.macrotrends.net/stocks/charts/TEAM/atlassian/revenue. Zugriff: 21. März 2019.

[9] Systems and software engineering – System life cycle processes. Standard, International Organization for Standardization, Geneva, CH, May 2015.

[10] Spartez Software sp. z o.o. sp.k. Agile Cards for Jira. https://spartez.com/products/agile-cards-for-jira. Zugriff: 23. März 2019.

[11] Thomas Alfie. Truffle — Easily synchronise physical and electronic backlogs. http://truffler.org/getting_started/. Zugriff: 23. März 2019.

[12] SMART Technologies ULC. Interaktive Displays für Unternehmen. https://home.smarttech.com/interactive-displays-for-business. Zugriff: 23. März 2019.

[13] Mike Cohn. *Succeeding with Agile Software Development using Scrum*. Person, Cambridge, 2010.

[14] Inc. Indiegogo. VoCore: A coin-sized Linux computer with wifi. https://www.indiegogo.com/projects/vocore-a-coin-sized-linux-computer-with-wifi#/. Zugriff: 25. März 2019.

[15] VoCore Studio. VoCore - The Coin-sized Linux Computer. https://vocore.io/v2.html. Zugriff: 25. März 2019.

[16] Felix Pfeifer. VoCore: Einplatinenrechner in Größe eines Zwei-Euro-Stücks. In *https://www.heise.de/make/meldung/VoCore-Einplatinenrechner-in-Groesse-eines-Zwei-Euro-Stuecks-3162336.html*. Zugriff: 25. März 2019.

[17] VoCore Studio. VoCore2 Firmware: Compile from Source. http://vonger.cn/?p=4400. Zugriff: 25. März 2019.

[18] The Rapsberry Pi Foundation. Raspberry Pi Zero W. https://www.raspberrypi.org/products/raspberry-pi-zero-w/. Zugriff: 25. März 2019.

[19] Arduino CC. ARDUINO UNO REV3. https://store.arduino.cc/arduino-uno-rev3. Zugriff: 25. März 2019.

[20] Atmel Corporation. *8-bit AVR Microcontroller with 32K Bytes In-System Programmable Flash Datasheet.* San José. http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7810-Automotive-Microcontrollers-ATmega328P_Datasheet.pdf, Zugriff: 25. März 2019.

[21] LTD. ESPRESSIF SYSTEMS (SHANGHAI) CO. ESP8266EX Datasheet. https://www.espressif.com/sites/default/files/documentation/0a-esp8266ex_datasheet_en.pdf. Zugriff: 25. März 2019.

[22] NodeMCU Team. NodeMCU Examples. https://www.nodemcu.com/index_en.html#fr_5475f7667976d8501100000f. Zugriff: 25. März 2019.

[23] Hitachi. HD44780U (LCD-II)(Dot Matrix Liquid Crystal Display Controller/Driver). https://cdn-shop.adafruit.com/datasheets/HD44780.pdf. Zugriff: 25. März 2019.

[24] E Ink Corporation. Electronic Ink. https://www.eink.com/electronic-ink.html. Zugriff: 02. April 2019.

[25] NXP Semiconductors. MFRC522 - Standard performance MIFARE and NTAG frontend. https://www.nxp.com/docs/en/data-sheet/MFRC522.pdf. Zugriff: 02. Juni 2019.

[26] Texas Instruments. Application Report - Bluetooth low energy Beacons. http://www.ti.com/lit/an/swra475a/swra475a.pdf, 2015. Zugriff: 02. Juni 2019.

[27] ArudinoJson: Efficient JSON serialization for embedded C++. Benoît Blanchon. https://arduinojson.org. Zugriff: 06. Mai 2019.

[28] Atlassian. JIRA Server platform REST API reference. https://docs.atlassian.com/software/jira/docs/api/REST/8.2.1/. Zugriff: 06. Mai 2019.

[29] Inc. npm. npm: axois. https://www.npmjs.com/package/axios. Zugriff: 29. Juni 2019.

[30] Atlassian. JIRA Service Desk REST API Reference. https://docs.atlassian.com/jira-servicedesk/REST/4.2.2/. Zugriff: 29. Juni 2019.

[31] Atlassian. JIRA Server platform REST API reference. https://docs.atlassian.com/software/jira/docs/api/REST/8.1.2/. Zugriff: 29. Juni 2019.

[32] Atlassian. Adding a custom field. https://confluence.atlassian.com/adminjiraserver/adding-a-custom-field-938847222.html. Zugriff: 06. Mai 2019.

[33] Susan C Hill, Joseph Jelemensky, and Mark R Heene. Queued serial peripheral interface for use in a data processing system (Patent US4816996), Mar 1989.

[34] Miguel Balboa. Arduino RFID Library for MFRC522. https://github.com/miguelbalboa/rfid. Zugriff: 06. Juni 2019.

[35] Waveshare Inc. 4.2inch e-Paper Module User Manual. https://www.waveshare.com/w/upload/2/20/4.2inch-e-paper-module-user-manual-en.pdf. Zugriff: 26. Juni 2019.

[36] Dan Radigan. JQL: the most flexible way to search Jira. https://www.atlassian.com/blog/jira-software/jql-the-most-flexible-way-to-search-jira-14. Zugriff: 28. Juni 2019.