Bachelorarbeit am Institut für Informatik der Freien Universität Berlin,

Arbeitsgruppe Software Engineering

# Phragile – Entwicklung einer Webanwendung für Sprint-Übersichten bei Wikimedia Deutschland

Jakob Warkotsch
Matrikelnummer: 4587134
j.warkotsch@gmail.com

Betreuer: Prof. Dr. Lutz Prechelt
Zweitgutachterin: Prof. Dr. Claudia Müller-Birn
Betreuer bei Wikimedia Deutschland: Abraham Taherivand

Berlin, 12.03.2015

**Zusammenfassung**

Regelmäßige Inspektionen und Messungen des Entwicklungsfortschritts sind ein fester Bestandteil der effektiven Anwendung von Scrum. Aussagekräftige Visualisierungen von Sprint-Daten sollen diesen Prozess unterstützen und den Zeitaufwand auf ein Minimum beschränken.
Bis Oktober 2014 setzte Wikimedia für Problemberichte und interne Aufgabenverteilung Bugzilla mit einer Scrum-Erweiterung ein. Durch einen Beschluss der Wikimedia Foundation in San Francisco wird seither Phabricator, eine Plattform für kollaborative Softwareentwicklung, verwendet. Da Phabricator keine Unterstützung für Scrum bietet, entstand der Bedarf nach einer Erweiterung um Sprints und entsprechende Übersichten für die Scrum Teams.
Diese Arbeit beschreibt die Entwicklung einer Applikation zur dynamischen Generierung von Sprint-Übersichten aus Phabricator-Projektdaten. In enger Zusammenarbeit mit der Softwareentwicklungsabteilung von Wikimedia Deutschland wurde eine Webanwendung entwickelt, die über Phabricators Webschnittstelle Sprint-Daten bezieht und daraus unter anderem Burndown-Diagramme, ein interaktives Sprint Backlog und weitere Visualisierungen erstellt.

## 0.1 Eidesstattliche Erklärung

Ich versichere hiermit an Eides Statt, dass diese Arbeit von niemand anderem als meiner Person verfasst worden ist. Alle verwendeten Hilfsmittel wie Berichte, Bücher, Internetseiten oder ähnliches sind im Literaturverzeichnis angegeben, Zitate aus fremden Arbeiten sind als solche kenntlich gemacht. Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungskommission vorgelegt und auch nicht veröffentlicht.

12.03.2015

Jakob Warkotsch

# Inhaltsverzeichnis


0.1 Eidesstattliche Erklärung . . . . . i

**1 Einleitung** **1**
1.1 Über Wikimedia Deutschland . . . . . 1
1.2 Problemstellung: Wechsel von Bugzilla und Scrumbugs zu Phabricator . . . . . 1
1.3 Ziel: Sprint-Übersichten für Phabricator . . . . . 2

**2 Scrum** **3**
2.1 Scrum Team . . . . . 3
2.2 Ereignisse . . . . . 3
2.3 Artefakte . . . . . 4
2.4 Scrum-Werkzeuge . . . . . 5

**3 Anforderungserhebung** **7**
3.1 Agile Methoden und Werkzeuge bei Wikimedia Deutschland . 7
3.2 Anforderungen an ein Scrum-Werkzeug bei Wikimedia Deutschland . . . . . 8
3.3 Untersuchung von Scrumbugs . . . . . 9
3.4 Untersuchung von Phabricator . . . . . 10
3.4.1 Integriert oder als eigenständige Applikation . . . . . 11
3.4.2 Authentifizierung . . . . . 13
3.4.3 Phabricator OAuth . . . . . 14
3.4.4 Conduit . . . . . 16

**4 Entwurf und Konzeption** **20**
4.1 UI Entwürfe . . . . . 20
4.1.1 Ablauf . . . . . 20
4.1.2 Paper Prototype . . . . . 22
4.1.3 Clickdummy . . . . . 23
4.2 Architekturentwürfe und -entscheidungen . . . . . 24

**5 Funktionsumfang Phragile** **26**
5.1 Authentifizierung . . . . . 26
5.2 Projektliste . . . . . 26
5.3 Projekte erstellen . . . . . 27
5.4 Sprint-Übersichten . . . . . 28
5.4.1 Burndown-Diagramm . . . . . 29
5.4.2 Sprint-Backlog-Tabelle . . . . . 30
5.4.3 Status-Kreisdiagramm . . . . . 30
5.5 Sprints erstellen . . . . . 31
5.6 Snapshots . . . . . 32
5.7 Conduit-Zertifikat . . . . . 32

**6 Entwicklung** **34**
6.1 Technologien . . . . . 34
6.2 Tests . . . . . 35
6.2.1 Akzeptanztests . . . . . 35
6.2.2 Unit-Tests . . . . . 36
6.3 Ablauf . . . . . 36
6.4 Hindernisse . . . . . 38
6.4.1 Phabricator . . . . . 39
6.4.2 Conduit . . . . . 39
6.4.3 libphutil . . . . . 41
6.4.4 OAuth . . . . . 41

**7 Fazit** **43**
7.1 Ausblick . . . . . 43
7.2 Resultat . . . . . 44

# 1 Einleitung

## 1.1 Über Wikimedia Deutschland

> “ Wikimedia ist eine weltweite Bewegung, die sich für die Idee des Freien Wissens einsetzt. Alle Wikimedia-Projekte – wie zum Beispiel Wikipedia – werden von der gemeinnützigen Wikimedia Foundation (Sitz in San Francisco, USA) betrieben. Über 40 unabhängige Länderorganisationen unterstützen die Wikimedia-Projekte vor Ort. Wikimedia Deutschland ist die älteste und größte von ihnen, wurde 2004 als gemeinnütziger Verein gegründet und hat heute mehr als 11.000 Mitglieder.” [wikb]

Federführend wird die Open-Source-Software Wikibase[1], welche die Grundlage für die freie Wissensdatenbank Wikidata[2] ist, seit 2012 im Entwicklungszentrum in Berlin umgesetzt. Für das Entwicklungsteam, das teils in der Berliner Geschäftsstelle, teils extern arbeitet, stellen agile Methoden und gute Werkzeuge einen essentiellen Bestandteil der Entwicklungsprozesse dar.

## 1.2 Problemstellung: Wechsel von Bugzilla und Scrumbugs zu Phabricator

Bis Oktober 2014 wurde sowohl für Problemberichte von Nutzern der Wikimedia Software, als auch intern für die Aufgabenverteilung des Entwicklerteams das von der Mozilla Foundation[3] entwickelte Bugzilla[4] verwendet. Unterstützend dazu war Scrumbugs[5] im Einsatz, eine Erweiterung von Bugzilla, welche dessen Daten um eine für Scrum ausgerichtete Oberfläche mit Burndown-Diagrammen und weiteren Statistiken ergänzt. Durch einen Beschluss der Wikimedia Foundation[6] wurden Bugzilla und Scrumbugs im Oktober 2014 durch Phabricator[7] abgelöst. Dieses soll durch eine benutzerfreundlichere Oberfläche mehr Nutzer und Ehrenamtliche dazu ermutigen, Problemberichte einzureichen und Entwicklern durch bessere Integration mit anderen Diensten wie Code-Hosting, Code-Reviews und Projektverwaltung den Arbeitsfluss optimieren [PAU14].
Eine Untersuchung von Phabricator durch die Softwareentwickler von Wikimedia Deutschland hat allerdings ergeben, dass wichtige Features – wie Sprint-Übersichten, Burndown-Diagramme und weitere Informationen zu den Aufgaben eines Sprints – fehlen.

[1] `http://wikiba.se`
[2] `http://www.wikidata.org/wiki/Wikidata:Main_Page`
[3] `https://www.mozilla.org/en-US/foundation/`
[4] `https://www.bugzilla.org`
[5] `https://scrumbu.gs`
[6] `https://wikimediafoundation.org/wiki/Home`
[7] `http://phabricator.org`

## 1.3 Ziel: Sprint-Übersichten für Phabricator

Ziel meiner Arbeit war die Erstellung einer Applikation, die für Scrum Master und Product Owner von Wikimedia Deutschland informative Sprint-Übersichten mit Burndown- und Statusdiagrammen, sowie einem interaktiven Sprint-Backlog erzeugt. Es wurde eine Webanwendung entwickelt, die über Phabricators Webschnittstelle die entsprechenden Projektdaten bezieht und es Benutzern erlaubt, Projekte und Sprints zu erstellen und eine Visualisierung der Projektdaten einzusehen. Die Anwendung wurde Phragile, als Wortspiel aus *"Phabricator, agile"*, genannt. Die erstellte Anwendung hat nicht den Anspruch alle Funktionalitäten von Scrumbugs abzudecken, sondern ist eine auf Wikimedia Deutschlands Bedürfnisse zugeschnittene Lösung zur benutzerfreundlichen Inspektion von Sprint-Daten.

# 2 Scrum

Da Scrum ein zentrales Thema dieser Arbeit ist, werden im Folgenden zunächst die Grundprinzipien und -begriffe von Scrum anhand des offiziellen "Scrum Guide"[8] [SS11] erklärt. Anschließend wird eine Studie zur Benutzung agiler Werkzeuge im Kontext dieser Arbeit betrachtet und ausgewertet.

**Definition**
Scrum ist ein Prozessframework, das seit den frühen 1990er Jahren für die Entwicklung komplexer Produkte eingesetzt wird. Es besteht aus einem Scrum Team, Ereignissen, Artefakten und Regeln, die alle einem bestimmten Zweck der erfolgreichen Anwendung von Scrum dienen [SS11].
Zum Zweck der Optimierung der Berechenbarkeit und der Risikokontrolle wird ein iterativer, inkrementeller Ansatz zur regelmäßigen Fertigstellung benutzbarer Produkte verwendet.

## 2.1 Scrum Team

Das Scrum Team ist ein selbstorganisierendes, interdisziplinäres Team, das ohne die Hilfe Außenstehender in der Lage sein sollte, das fertige Produkt zu erstellen. Es besteht aus dem Product Owner, dem Entwicklungsteam und dem Scrum Master [SS11].

**Product Owner**
Der Product Owner ist für die Wertmaximierung des Produkts und der Arbeit des Entwicklungsteams zuständig. Er hat außerdem die Aufgabe, die Einträge des Product Backlogs zu verwalten und zu priorisieren [SS11].

**Entwicklungsteam**
Das Entwicklungsteam besteht in der Regel aus mindestens drei Personen, die selbst entscheiden, wie sie ihre Arbeit einteilen und verrichten [SS11].

**Scrum Master**
Der Scrum Master hat dafür zu sorgen, dass Scrum von allen Beteiligten verstanden und regelgerecht angewandt wird. Des Weiteren ist er für die Moderation der ein- und ausgehenden Kommunikation des Entwicklungsteams zuständig [SS11].

## 2.2 Ereignisse

Ereignisse in Scrum dienen der Minimierung des Bedarfs an außerplanmäßigen Meetings. Alle Ereignisse sind zeitbeschränkt und sollten nicht verlängert werden [SS11].

[8] `http://www.scrumguides.org/scrum-guide.html`

**Sprint**
Ein Sprint ist ein zeitlich beschränkter Entwicklungszyklus von einem Monat oder weniger in dem ein benutzbares Produkt erstellt wird [SS11].

**Sprint-Planung**
Bei der Sprint-Planung wird vom gesamten Scrum Team das Ziel des Sprints festgelegt und gegebenenfalls vom Entwicklungsteam der Plan zum Erreichen des Ziels entworfen [SS11].

**Daily Scrum**
Das Daily Scrum ist ein tägliches, 15-minütiges Treffen, in dem jedes Mitglied des Teams seinen Beitrag zum Erreichen des Sprint-Ziels der letzten 24 Stunden reflektiert und den Plan für die nächsten 24 Stunden und mögliche Hindernisse schildert [SS11].

**Sprint Review**
Das Sprint Review findet am Ende jedes Sprints statt und dient der Inspektion des Product Backlogs. Anhand der im Sprint fertiggestellten Aufgaben soll untersucht werden, was mögliche Aufgaben für den nächsten Sprint sein können und was ins Product Backlog aufgenommen werden muss [SS11].

**Sprint Retrospektive**
Die Sprint Retrospektive dient der Verbesserung von zwischenmenschlichen Beziehungen, Prozessen und Werkzeugen. Es soll identifiziert werden, was im vergangenen Sprint gut funktioniert hat, wo es Optimierungsbedarf gibt und wie entsprechende Änderungen umgesetzt werden können [SS11].

## 2.3 Artefakte

**Product Backlog**
Das Product Backlog ist eine sortierte Liste mit Einträgen von allen möglichen Anforderungen an das Produkt oder möglichen Änderungen am Produkt. Diese Einträge sollten eine Beschreibung, eine Umfangsabschätzung, sowie eine Einschätzung des Wertes beinhalten [SS11].

**Sprint Backlog**
Das Sprint Backlog enthält alle Funktionalitäten, die das Entwicklungsteam für die nächste Version des Produkts implementieren wird. Es enthält alle Einträge aus dem Product Backlog, die für diesen Sprint ausgewählt wurden [SS11].

**User Stories**
User Stories sind eine Variante, wie Einträge im Sprint- und Product Backlog beschrieben werden. Die Beschreibung einer Funktionalität erfolgt aus der Sicht einer Person, für die diese Funktionalität nützlich ist. Diese folgen im Englischen oft der Form "As a [type of user], I want [some goal] so that [some reason]." [Sofb].

**Burndown-Diagramm**
Ein Burndown-Diagramm ist eine mögliche Veranschaulichung des Entwicklungsfortschritts eines laufenden Sprints. Hierbei wird die Menge der verbleibenden Arbeit (oft in Story Points oder Arbeitsstunden) auf der Y-Achse über einen Zeitabschnitt – in der Regel über die Dauer eines Sprints – abgebildet. Es entsteht somit ein Graph, dessen Punkte angeben, wie viel Arbeit das Entwicklungsteam zu einem bestimmten Zeitpunkt zum Erreichen des Sprint-Ziels noch verrichten muss[Sofa].

## 2.4 Scrum-Werkzeuge

2011 wurde von der Firma Ericsson die Studie "Survey of Agile Tool Usage and Needs" [AMKM11] zur Benutzung agiler Werkzeuge in über 120 Unternehmen aus 35 verschiedenen Ländern durchgeführt. Hierbei wurden den Befragten 10 Fragen zu genutzten agilen Methoden, Art der Entwicklungsteams (verteilt oder lokal), Werkzeugen und Wertung dieser Werkzeuge vorgelegt.
Es wurde festgestellt, dass Scrum mit 53% die am häufigsten verwendete agile Methode ist, gefolgt von XP in Kombination mit Scrum (35%). Die Liste der verwendeten Werkzeuge wird mit 26% von Papier und Tafeln (Whiteboards, Pinnwände, Notizzettel o.ä.) und Spreadsheets (23%) angeführt. Obwohl kein konkretes Software-Werkzeug auf einen entsprechend hohen Anteil kam, benutzen doch 65% aller befragten Unternehmen eine oder mehrere der Softwarelösungen für agile Entwicklungsmethoden. Eine interessante Beobachtung ist, dass bei verteilten Teams der Anteil mit über 85% wesentlich höher ist. Bei der Wertung der Tools sticht hervor, dass mit Abstand am häufigsten gute Benutzbarkeit als Vorteil der verwendeten Tools genannt wurde. Gleichzeitig haben 93% einen oder mehrere negative Aspekte ihrer verwendeten Werkzeuge erwähnt.

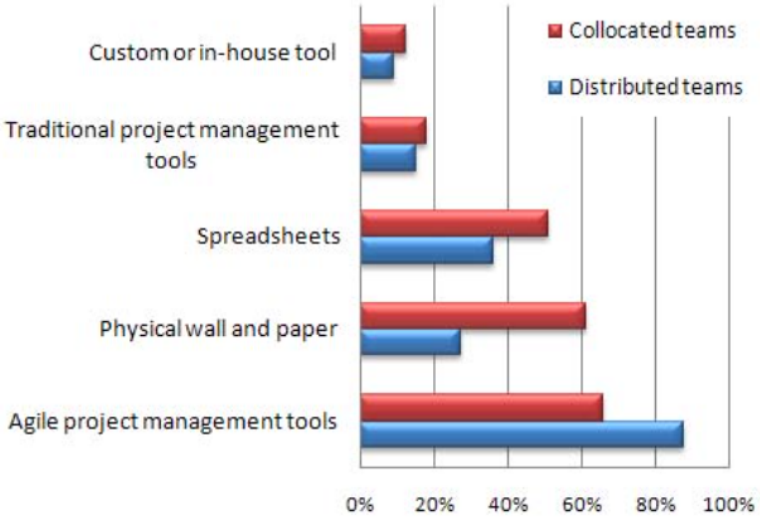



Abbildung 1: Benutzung verschiedener Werkzeuge in verteilten und lokalen Teams[AMKM11]

Anhand dieser Studie kann ich für meine Arbeit folgern, dass unterstützende Software für agile Methoden besonders bei verteilten Entwicklungsteams wie dem von Wikidata eine wichtige Rolle spielt. Des Weiteren werde ich für Phragile verschiedene Methoden zur Optimierung der grafischen Oberfläche einsetzen um eine gute Benutzbarkeit zu erzielen und hierzu häufig Rückmeldung von Wikidatas Scrum Master und anderen Endbenutzern erfragen.

# 3 Anforderungserhebung

In diesem Kapitel werde ich den Ablauf der Anforderungserhebung schildern. Als Ausgangspunkt werden zunächst der Gebrauch von agilen Methoden bei Wikimedia Deutschland und die Anforderungen der Hauptbenutzer erläutert. Anschließend werden diese mit der zuvor verwendeten Software, Scrumbugs, und der Funktionalität von Phabricator abgeglichen.

## 3.1 Agile Methoden und Werkzeuge bei Wikimedia Deutschland

Agile Entwicklungsprozesse sind ein fester Bestandteil der Softwareentwicklung bei Wikimedia Deutschland. Insbesondere das Wikidata-Team arbeitet seit 2012 mit agilen Methoden. Neu gewonnene Erkenntnisse, Anfragen und Bedarfsanalysen aus der Open-Source-Community müssen dabei im Entwicklungsprozess für neue Funktionen Berücksichtigung finden. Bei der Realisierung größerer Projekte wird Scrum angewandt, aber auch einige Techniken aus XP[9] und Kanban[10] wie z. B. testgebtriebene Entwicklung, Paarprogrammierung und Visualisierung und Messung des Arbeitsflusses kommen zum Einsatz. Diese werden durch verschiedene analoge und digitale Werkzeuge unterstützt.
Zum Verständnis der Arbeitsprozesse und bestimmter Methoden habe ich mir zu Beginn der Arbeit mehrere Treffen wie die Daily Scrums oder Sprint-Planung vor Ort angesehen. Die Inspektion der Arbeit erfolgt bei Wikimedia Deutschlands Entwicklungsteams während der Daily Scrums auf Whiteboards mit den Spalten “To Do”, “Doing”, “In Review” und “Done”, die den Arbeitsstatus der Aufgaben des laufenden Sprints abbilden.

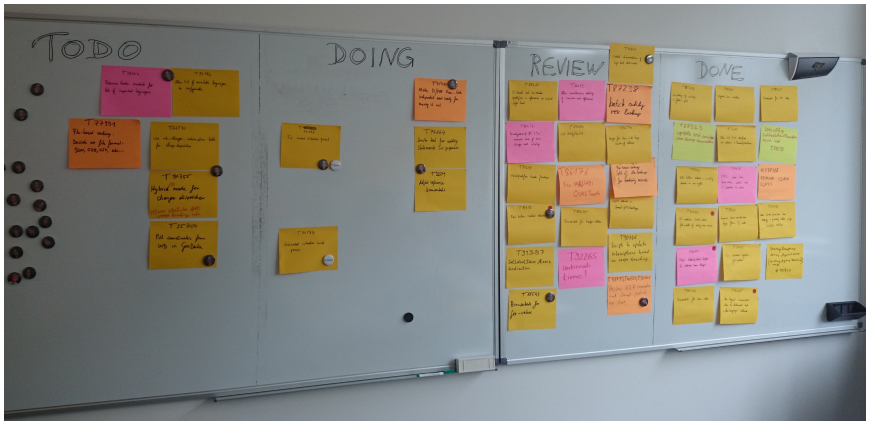



Abbildung 2: Die phyische Kanban-Tafel des Wikidata-Teams

[9] http://www.extremeprogramming.org/rules.html
[10] http://www.kanban-plakat.de

Analog dazu ist ein entsprechendes Sprint Backlog online abrufbar, das für jede Aufgabe, neben dem Status und der Beschreibung, eine Komplexitätsabschätzung in Story Points enthält. Hierfür wurde bis Oktober 2014 Scrumbugs und seitdem Phabricator verwendet.
Die Sprint-Übersichten des jeweiligen Werkzeugs dienen zum einen der täglichen Einsicht über den Fortschritt der Entwicklung des laufenden Sprints und zum anderen zur Umfangsabschätzung des Sprint-Ziels bei der Sprint-Planung.

## 3.2 Anforderungen an ein Scrum-Werkzeug bei Wikimedia Deutschland

Vom Scrum Master und der Produktmanagerin des Wikidata-Teams wurde eine Liste der wichtigsten Anforderungen an das Scrum-Werkzeug erstellt. Um die gewünschte Funktionalität des neuen Werkzeug zu erkennen und den Ist- und Sollzustand der bisher verwendeten Softwarelösungen darzustellen, werden im Folgenden die Anforderungen tabellarisch dargestellt und mit Indikatoren über die Verfügbarkeit in Scrumbugs (**SB**) bzw. Phabricator (**PH**) versehen.

| **Funktionalität** | **SB** | **PH** |
|---|---|---|
| Sprints definieren | ✓ | ✓ |
| Zeitfenster mit Start- und Enddatum für Sprints festlegen | ✓ | - |
| Tasks Story Points zuweisen | ✓ | ✓ |
| Sprints Tasks zuweisen und wieder entfernen | ✓ | ✓ |
| Anzeige der Gesamtzahl der Story Points eines Sprints | ✓ | - |
| Anzeige der Anzahl der verbleibenden Story Points eines Sprints | ✓ | - |
| Burndown-Diagramm | ✓ | - |
| Übersicht vergangener Sprints einsehen | - | - |
| Anzahl der geschlossenen Story Points an einem bestimmten Tag sehen | ✓ | - |
| Status aller Tasks angezeigen | ✓ | ✓ |
| Komponente eines Tasks sehen | ✓ | ✓ |
| Spalten im Workboard definieren (To Do, Doing, In Review, Done) | - | ✓ |

Tabelle 1: Funktionalität für den Scrum Master

| **Funktionalität** | **SB** | **PH** |
|---|---|---|
| Product Backlog einsehen | ✓ | ✓ |
| Prioritäten für Tasks festlegen | ✓ | ✓ |
| Tasks mit Schlüsselwörtern versehen (z. B. needs discussion, next sprint, etc) | ✓ | ✓ |
| User Stories erstellen | ✓ | - |

Tabelle 2: Funktionalität für den Product Owner

Eine gute Scrum-Erweiterung für Phabricator sollte also möglichst alle Funktionalitäten abdecken, die in der **PH**-Spalte nicht mit einem ✓versehen wurden. Das Hauptaugenmerk dieser Arbeit lag auf der Berücksichtigung der Anforderungen des Scrum Masters, sodass die Funktionalität "User Stories erstellen", welche für den Product Owner eine höhere Relevanz hat, als optional markiert wurde.

## 3.3 Untersuchung von Scrumbugs

Scrumbugs ist eine Scrum-Erweiterung für den von Mozilla entwickelten Bugtracker, Bugzilla. Es erlaubt eingeloggten Benutzern das Erstellen von Teams, Projekten und Sprints. Über Bugzillas Webschnittstelle werden Tasks[11] für ein Projekt bezogen und in das Product Backlog übertragen, aus dem sie dann nach Erstellung eines Sprints in ein Sprint Backlog überführt werden können [scr].
Für Sprints können anschließend Übersichtsseiten angezeigt werden, die das Sprint Backlog als sortierbare Tabelle mit Informationen zu den Aufgaben des Sprints enthalten. Des Weiteren werden Visualisierungen wie das Burndown-Diagramm und verschiedene Kreisdiagramme generiert und angezeigt.

[11] Task: Teilaufgabe eines Eintrags im Product Backlog

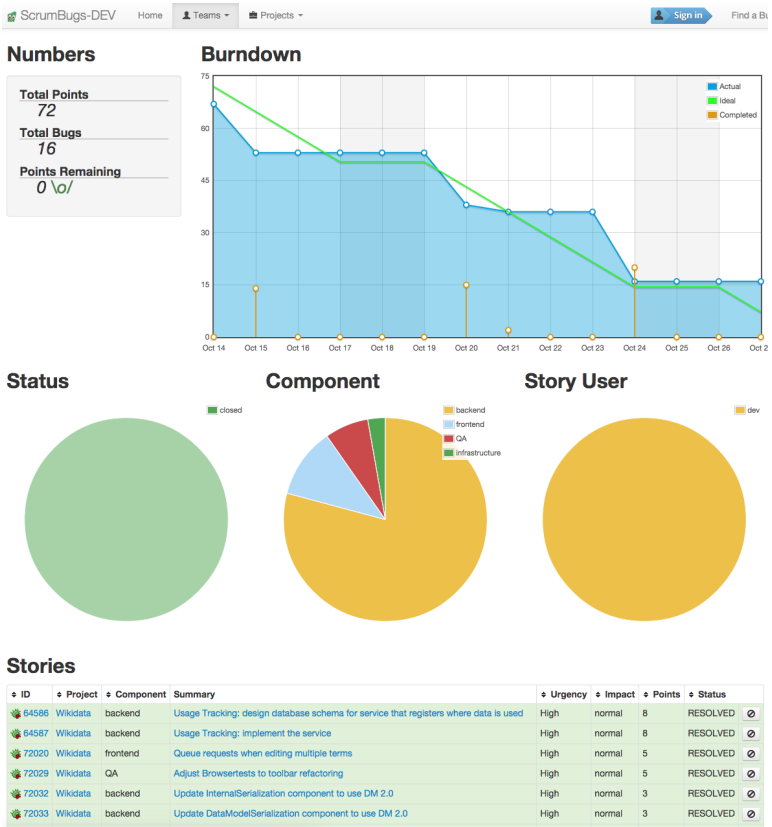



Abbildung 3: Eine Sprint-Übersicht von Wikidata in Scrumbugs[wika]

Scrumbugs war ein essentieller Bestandteil der Entwicklungsprozesse und wurde von Wikidatas Scrum Team täglich verwendet.

## 3.4 Untersuchung von Phabricator

Phabricator ist eine Zusammenstellung freier Webapplikationen zur kooperativen Softwareentwicklung. Es enthält unter anderem Komponenten für Bugtracking, Code-Reviews, Code-Hosting und Dokumentation [phag].
Scrum wird von Phabricator nicht explizit unterstützt, es kann jedoch dafür genutzt werden. Durch die Erstellung entsprechend benannter Projekte können Sprints abgebildet werden. Ein Task kann gleichzeitig mehreren Projekten angehören, sodass ein Product Backlog durch die Zuweisung von Tasks zu einem Hauptprojekt dieses Produkts gehalten werden kann. Bei der Sprint-Planung werden Tasks dieses Sprints vom Hauptprojekt in das Sprint-Projekt verschoben.
Tasks enthalten bei einer neu aufgesetzten Installation von Phabricator kein Feld für Story Points. Dies kann jedoch über die Konfiguration des Bugtrackers ergänzt werden, die das Hinzufügen beliebiger benutzerdefinierter

Felder erlaubt [phad].
Visualisierungen für den Fortschritt des Teams in Sprints wie bei Scrumbugs' Sprint-Übersichten werden von Phabricator nicht erzeugt.

### 3.4.1 Integriert oder als eigenständige Applikation

Eine wichtige Aufgabe, die es vor Beginn der Implementierung zu lösen galt, bestand darin zu erörtern, ob Phragile in Phabricator integriert oder als eigenständige Applikation entwickelt werden sollte. Die Argumente für und gegen Integration von Phragile in Phabricator werden im Folgenden erläutert.

**Phragile in Phabricator integriert**
Eine Integration in Phabricator bedeutet, dass Phragiles Sprint-Übersichten als Unterseite von Phabricator direkt aus der Projektansicht aufrufbar wären.

**Vorteile**

- Eine Schnittstelle, sowie entsprechende PHP-Interfaces für die Entwicklung integrierter Applikationen sind bereits vorhanden und werden von Phabricator selbst intern genutzt [phab].
- Die Integration verbessert die User Experience durch weniger Kontextwechsel beim Anlegen und Einsehen von Sprints.

**Nachteile**

- Integrierte Applikationen werden in absehbarer Zeit nicht unterstützt. Eine offizielle Anmerkung dazu lautet: "Internal Applications: Phabricator might be a suitable platform for development of internal (your company only) applications in 1-2 years."[phab]
- Wikimedias Phabricator-Instanz wird von der WMF in Amerika betreut. Der Einfluss von Wikimedia Deutschland auf Deployments und spezielle Änderungen ist deshalb eingeschränkt.
- Der Code liefe auf demselben Server wie Phabricator und könnte im schlimmsten Fall (z. B. wegen Inkompatibilität nach einem Upgrade) zum Versagen wichtiger Dienste führen.

**Phragile als eigenständige Applikation**
Die zweite Möglichkeit bestand in der Entwicklung einer externen Applikation, die Informationen über Phabricators Webschnittstelle bezieht.

### Vorteile

- Eine entsprechende Webschnittstelle ist vorhanden und dokumentiert[12].
- Phabricator hat einen OAuth[13]-Dienst[14] zur Authentifizierung seiner Benutzer auf externen Applikationen eingebaut.
- Für die Schnittstelle, sowie den OAuth-Client stellt Phabricator eine Bibliothek namens *libphutil*[15] zur Verfügung.
- Eine externe Implementierung ist nicht von Phabricator-Code abhängig und vereinfacht Dritten die Weiterverwendung.
- Die Bereitstellung der Phragile-Instanz ist nicht von der Wikimedia Foundation in Amerika abhängig und müsste von dieser auch nicht geprüft oder betreut werden.

### Nachteile

- Eine externe Applikation von Grund auf zu entwickeln erfordert höchstwahrscheinlich mehr Programmieraufwand, da weder bestehende Oberflächenelemente, noch interne Schnittstellen und Module von Phabricator weiterverwendet werden können.
- Phabricators Webschnittstelle befindet sich in aktiver Entwicklung und häufige Änderungen sind möglich.
- Die OAuth-Funktionalität zur Authentifizierung von registrierten Phabricator-Nutzern befindet sich noch im *Prototype*-Zustand, was bedeutet, dass sie noch nicht offiziell Unterstützt wird und häufige Änderungen wahrscheinlich sind [phaf].

Anhand dieser Argumente erschien die zweite Variante als die bessere Lösung, da vor allem die mangelnde Unterstützung für integrierte Applikationen einen großen Nachteil und die Freiheiten bei Änderungen, Betreuung und Bereitstellung einer alleinstehenden Anwendung einen signifikanten Vorteil darstellten.

[12] `https://secure.phabricator.com/book/phabdev/article/conduit/`
[13] Ein standardisiertes Authorisierungsprotokoll `http://oauth.net`
[14] `https://secure.phabricator.com/book/phabcontrib/article/using_oauthserver/`
[15] `https://github.com/phacility/libphutil`

### 3.4.2 Authentifizierung

Da Benutzer von Phragile unter anderem in der Lage sein sollen Sprints anzulegen und zu verwalten, wurde ein Authentifizierungsmechanismus notwendig um die Applikation gegen etwaige böswillige oder unachtsame Benutzer abzusichern. Hierfür kamen entweder ein eigenes Benutzersystem für Phragile oder die Implementierung einer OAuth-Lösung in Frage.

**OAuth**
OAuth ist ein Protokoll, das es Anbietern erlaubt, bestimmte Aktionen von einer entfernten Anwendung authorisiert auszuführen ohne dabei das Benutzerpasswort weiterzugeben, nachdem eine entsprechende Bestätigung des Benutzers erfolgt [oaua].
Hierfür habe ich mit Wikimedia Deutschlands Phabricator-Experten und dem Bereichsleiter der Softwareentwicklung mögliche Ansätze diskutiert. Drei Lösungsvorschläge kamen in die engere Auwahl:

a) Ein eigenes Benutzersystem für Phragile, das Registrierung beliebiger Konten ermöglicht, die nicht notwendigerweise mit Phabricator-Konten verknüpft sein müssen

b) Die Verwendung von MediaWiki OAuth[16], sodass sich Benutzer mit ihrem bestehenden MediaWiki[17] Benutzerkonto auf Phragile anmelden können

c) Phabricator OAuth Server, Phabricators eigenener Implementierung von OAuth, die es Phabricator-Nutzern ermöglichen würde, sich auf Phragile anzumelden

Möglichkeit *a)* wurde schnell verworfen, da es zum einen für die Benutzer mehr Aufwand darstellt und die Verwaltung von Benutzerkonten weitere Schwierigkeiten mit sich bringt.
*b)* und *c)* haben beide für Nutzer von Phragile den Vorteil, dass eine Registrierung eines neuen Benutzerkontos für die Benutzung von Phragile für Wikimedia-Mitarbeitende nicht nötig ist, da sowohl ein entsprechendes MediaWiki-Konto und ein Phabricator-Konto schon vorhanden sind. Zur Implementierung von *b)* würde außerdem keine weitere Konfiguration von Phabricator notwendig sein, während *c)* eine Aktivierung des OAuth-Server-Moduls voraussetzt, welches sich zu diesem Zeitpunkt noch in einem frühen Entwicklungsstadium befindet. Die Entscheidung fiel trotzdem auf Variante *c)*, da die direkte Verknüpfung von Phragile zu Phabricator die Verwendung dessen Benutzerkonten begünstigt und es außerdem Drittbenutzern die Möglichkeit bietet, Phragile mit Phabricator zu verwenden, auch wenn sie nichts mit MediaWiki oder Wikimedia zu tun haben.

[16] `http://www.mediawiki.org/wiki/Extension:OAuth`
[17] `http://www.mediawiki.org/wiki/MediaWiki`

Durch die Verwendung von Phabricators OAuth-Dienst können sich alle registrierten Benutzer dieser Phabricator-Instanz – unabhängig von dessen Authentifizierungsmechanismus – bei Phragile anmelden. Der Unterschied wird beim Vergleich der zwei folgenden Abbildungen deutlich.

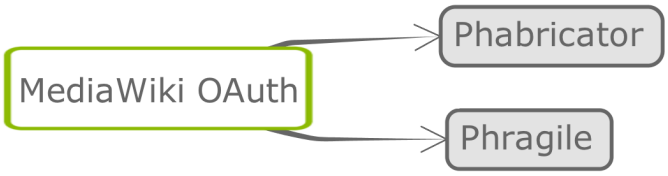



Abbildung 4: MediaWiki OAuth zur Authentifizierung auf Phabricator und Phragile

Abbildung 5: Beliebiger Auth.-Mechanismus für Phabricator und Phabricator OAuth zur Authentifizierung auf Phragile

### 3.4.3 Phabricator OAuth

Vorraussetzung für eine Implementierung entsprechend dieser Entscheidung waren zum einen Phabricators OAuth-Dienst und die Webschnittstelle. Um an diesen Stellen nicht zu einem fortgeschrittenem Zeitpunkt der Entwicklung auf Probleme zu stoßen, wurden diese anschließend auf Funktionstüchtigkeit untersucht.
OAuth bietet dem Benutzer die Möglichkeit, sich bei Phragile mit seinem Phabricator-Benutzerkonto anzumelden. Diese Funktionalität ist notwendig, damit Benutzer z. B. durch Phragile neue Sprints auf Phabricator anlegen kann.
Um diesen Mechanismus zu testen, wurde zunächst die OAuth-Server-Applikation in einer lokalen Phabricator-Instanz aktiviert und dort Phragile als neue Applikation registriert. Anschließend habe ich das folgende Skript entwickelt, was mithilfe manueller Benutzereingaben den OAuth Handshake durchführt. Tests dieser Art stellten für mich einen erheblichen Teil der Untersuchung verschiedener Funktionalitäten von Phabricator dar, da dessen Dokumentation, wie im Abschnitt 6.4 beschrieben, nicht immer hilfreich war.

Listing 1: Skript zur Überprüfung von Phabricators OAuth-Dienst

```
<?php

// load OAuth constants specific to the installation
include 'config.php';

echo 'Click the following link: ', PHABRICATOR_URL, '
    oauthserver/auth/?', http_build_query([
    'response_type' => 'code',
    'client_id'     => PHRAGILE_CLIENT_ID,
    'redirect_uri'  => PHRAGILE_URL,
]), "\n";

$authorization_code = readline('Follow the steps from the
    Phabricator dialogue. Then, after being redirected paste
     the `code` http query parameter here: ');

$access_token_url = PHABRICATOR_URL . 'oauthserver/token/?'
    . http_build_query([
    'client_id'     => PHRAGILE_CLIENT_ID,
    'client_secret' => PHRAGILE_CLIENT_SECRET,
    'code'          => $authorization_code,
    'grant_type'    => 'authorization_code',
    'redirect_uri'  => PHRAGILE_URL,
]);

// parse HTTP Response as JSON
$response = json_decode(file_get_contents($access_token_url)
    );

echo print_r(json_decode(file_get_contents(PHABRICATOR_URL .
     "api/user.whoami?api.access_token={$response->
    access_token}")));
```

Führt man das Skript auf der Kommandozeile aus, wird man aufgefordert, einem Link zu Phabricator zu folgen und anschließend einen Authorisierungscode einzufügen. Nach Bestätigung der Authorisierung, leitet Phabricator den Benutzer mit einem Authorisierungscode als HTTP-Query-Parameter auf die bei der Konfiguration hinterlegte URL weiter. Fügt man diesen Code nun auf der Kommandozeile ein, führt das Skript einen HTTP-Request zur Anfrage des Access-Tokens bei Phabricator aus. Nach Erhalt dieses Tokens wird zur Verifizierung eine API-Anfrage gestellt, die die Benutzerdaten wie folgt auf der Kommandozeile ausgibt.

Listing 2: Ausgabe des Skripts

```
$ php oauth_call.php
[... Dialog ...]
stdClass Object
(
    [result] => stdClass Object
        (
```

```
            [phid] => PHID-USER-pc3htyac6qkdruc7ezuy
            [userName] => Jakob
            [realName] => Jakob Warkotsch
            [image] => http://phabricator.dev/res/
               phabricator/3eb28cd9/rsrc/image/avatar.png
            [uri] => http://phabricator.dev/p/Jakob/
            [roles] => Array
                (
                    [0] => admin
                    [1] => verified
                    [2] => approved
                    [3] => activated
                )

        )

    [error_code] =>
    [error_info] =>
)
```

Dieses Skript diente reinen Testzwecken und versprach eine ausreichende Funktionalität des OAuth-Diensts. Allerdings ist es nicht möglich, über den OAuth-Dienst im Namen des Benutzers Schreibzugriffe wie das Erstellen von Sprints oder das Ändern von Tasks auszuführen. Die einzige über OAuth verfügbare Methode der Schnittstelle ist `user.whoami`.

> "There are only two scopes supported at this time. `offline_access` - allows an access token to work indefinitely without expiring. `whoami` - allows the client to access the results of Conduit.whoami on behalf of the resource owner." [oaub]

### 3.4.4 Conduit

Phragile soll nach Projekten gruppierte Sprint-Übersichten enthalten, die Informationen zu Tasks dieser Sprints visualisieren. Entsprechend müssen folgende Daten über Conduit, Phabricators Webschnittstelle, verfügbar sein:

a) Titel, Identifizierer, Start- und Enddatum aller Sprints eines Projekts

b) Titel, Identifizierer, Status und Priorität aller Tasks

c) Änderungsdatum jedes Status

Anhand einer Untersuchung der grafischen Web-Oberfläche von Conduit musste ich feststellen, dass *a)* nicht möglich ist. Phabricator kennt weder eine Unterscheidung zwischen Sprint und Projekt, noch sind Verschachtelungen von Projekten möglich. Die Felder, nach denen Sprints (d. h. Phabricator-Projekte) über die API mittels `project.query`[18] angefragt werden können, sind in der folgenden Abbildung dargestellt.

[18] `https://secure.phabricator.com/conduit/method/project.query/`

**project.query**

Description Execute searches for Projects.

Returns list

Errors **ERR-CONDUIT-CORE:** See error message for details.

Enter parameters using **JSON**. For instance, to enter a list, type:
`["apple", "banana", "cherry"]`

ids
optional list<int>

names
optional list<string>

phids
optional list<phid>

slugs
optional list<string>

status
optional string-constant<"status-any", "status-open", "status-closed", "status-active", "status-archived">

members
optional list<phid>

limit
optional int

offset
optional int

Abbildung 6: Felder der Conduit-Methode `project.query` in der Web-Oberfläche

Ein weiteres Problem stellten Start- und Enddatum dar. Man kann zwar beliebige Felder für Phabricator-Projekte hinzufügen, diese allerdings weder über die API lesen, noch beschreiben. Als Umgehung dieses Problems wurde nach Rücksprache mit dem Scrum Master des Wikidata-Teams beschlossen, dass Sprints stattdessen in Phragile angelegt werden sollten. Das Anlegen eines Sprints mit Titel, Start- und Enddatum würde entsprechend über Phragiles Oberfläche geschehen und zum einen über die Conduit-Methode `project.create`[19] den Sprint als Phabricator-Projekt erzeugen und gleichzeitig anhand dessen Rückgabewert den Phabricator-Identifizierer dieses Sprints in Phragiles eigener Datenbank zusammen mit Sprint-Titel, Start- und Enddatum speichern. Diese Entscheidung wurde in dem Bewusstsein getroffen, dass das Speichern von Daten eines Objekts an zwei Orten auch Komplikationen bei der Löschung, Synchronisation oder Editierung mit sich bringt.

[19] `https://secure.phabricator.com/conduit/method/project.create/`

Das Anfragen aller Tasks eines Sprints stellte kein derartiges Problem dar. Die Conduit-Methode `maniphest.query`[20] erlaubt das Anfragen aller nötigen Informationen zu Tasks eines Phabricator-Projekts anhand des Phabricator-Identifizierers dieses Sprints, der sich dank der Lösung von *a)* in Phragiles Datenbank befindet.
*c)* wurde für die Erstellung eines Burndown-Diagramms gebraucht. Alle Manipulationen von Tasks werden in Phabricator in sogenannten Transactions gespeichert, die über die Conduit-Methode `maniphest.gettasktransactions`[21] beziehbar sind. Eine JSON-Antwort der API auf `maniphest.gettasktransactions` sieht beispielsweise wie folgt aus:

Listing 3: Antwort auf maniphest.gettasktransactions

```
{
  "18": [
    {
      "taskID": "18",
      "transactionPHID": "PHID-XACT-TASK-qjsdxrqk7265xmp",
      "transactionType": "core:edge",
      "oldValue": [],
      "newValue": {
        "PHID-PROJ-t6lwggn3uztcagy2qulz": {
          "dst": "PHID-PROJ-t6lwggn3uztcagy2qulz",
          "type": 41,
          "data": []
        }
      },
      "comments": null,
      "authorPHID": "PHID-USER-q5f4243ipckimwuogawz",
      "dateCreated": "1417789433"
    },
    {
      "taskID": "18",
      "transactionPHID": "PHID-XACT-TASK-fhyn6ctvmlbfeci",
      "transactionType": "core:customfield",
      "oldValue": null,
      "newValue": "8",
      "comments": null,
      "authorPHID": "PHID-USER-q5f4243ipckimwuogawz",
      "dateCreated": "1417789433"
    },
    {
      "taskID": "18",
      "transactionPHID": "PHID-XACT-TASK-wc4rcpha2jeushy",
      "transactionType": "status",
      "oldValue": "open",
      "newValue": "resolved",
      "comments": null,
      "authorPHID": "PHID-USER-pc3htyac6qkdruc7ezuy",
```

[20] https://secure.phabricator.com/conduit/method/maniphest.query/
[21] https://secure.phabricator.com/conduit/method/maniphest.gettasktransactions/

```
      "dateCreated": "1418310335"
    }
  ]
}
```

Für das Burndown-Diagramm ist nur interessant, wann ein Task geschlossen wurde. Das kann herausgefunden werden, indem man die "dateCreated"-Eigenschaft der letzten Transaction betrachtet, die `"transactionType": "status"` beinhaltet und als `oldValue` einen nicht-geschlossenen (d. h. nicht "wontfix" oder "resolved") und als `newValue` einen geschlossenen Wert hat. Im Beispiel beschreibt die letzte Transaction eine Schließung dieses Tasks am 11.12.2014.

# 4 Entwurf und Konzeption

Im Folgenden werde ich erläutern, wie Entscheidungen bezüglich der Benutzeroberfläche und der Architektur getroffen wurden. Zusammen mit Wikidatas Scrum Master wurden von mir nach verschiedenen Methoden erstellte Entwürfe für das User Interface ausgewertet und angepasst. Es wurde außerdem mit Wikimedias Softwarearchitekt eine mögliche Architektur skizziert und diskutiert.

## 4.1 UI Entwürfe

Durch das Erstellen von Skizzen und Prototypen der grafischen Oberfläche lassen sich konkrete Vorstellungen des Entwicklers und des Kunden darstellen. Sie bieten eine Diskussionsgrundlage und können frühzeitig Missverständnisse aufdecken. Durch schnelles Erzeugen und Verändern dieser Skizzen können auf dem Papier oder am Whiteboard in kurzer Zeit neue Ideen visualisiert werden, ohne Code schreiben zu müssen oder ein großes Grafikprogramm zu starten [GCMB11].
Für Phragile wurden drei verschiedene Entwurfsmethoden angewandt. Es wurde eine Skizze aller Hauptelemente der Applikation mithilfe der "Sequential Storyboard"-Methode [GCMB11] zum Verständnis des Arbeitsablaufes erstellt, ein Paper Prototype der Sprint-Übersicht, um verschiedene Anordnungen der Seitenelemente zu erwägen und ein interaktiver Prototyp ("Clickdummy") um Interaktionsmöglichkeiten zu testen.

### 4.1.1 Ablauf

Um ein gemeinsames Verständnis für den Ablauf einer Benutzung der Applikation zu bekommen, habe ich zunächst auf Papier die Hauptseiten der Applikation einzeln skizziert und diese anschließend an ein Whiteboard geklebt und mit Annotationen versehen.

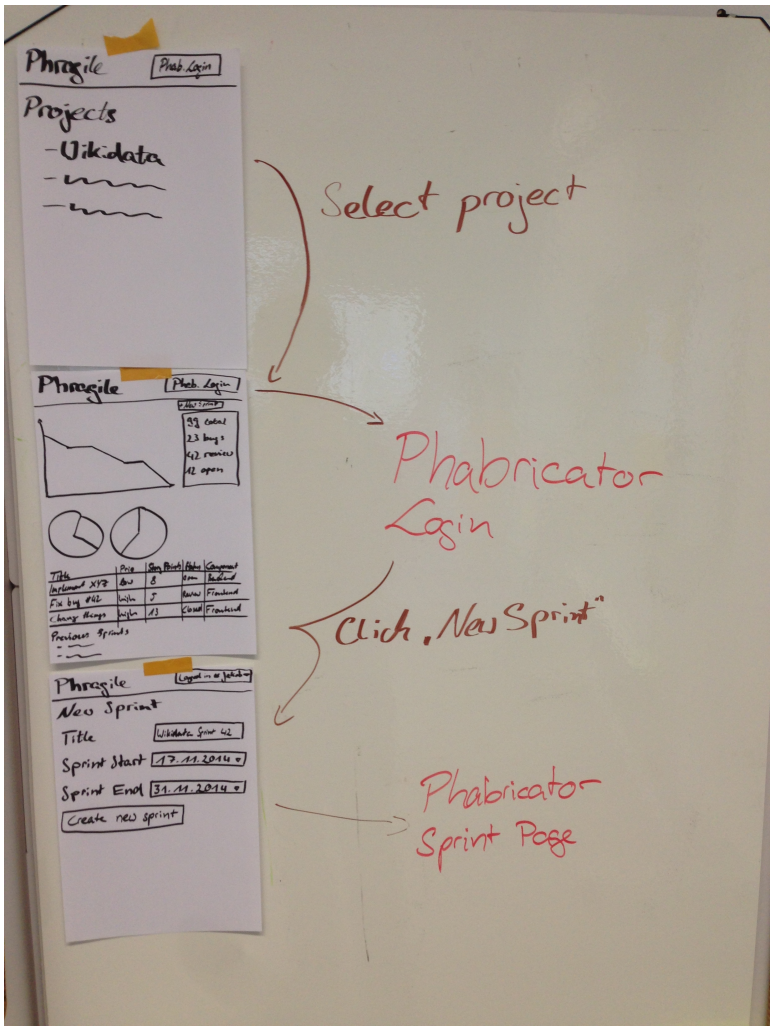



Abbildung 7: Ablauf in Phragile

Die erste Seite stellt die Startseite von Phragile mit einer Liste aller Projekte dar. Nach der Auswahl eines Projekts wird eine Übersicht des letzten Sprints dieses Projekts angezeigt. Im eingeloggten Zustand sieht der Benutzer dort einen "New Sprint"-Button zur Erstellung eines neuen Sprints. Das letzte Bild zeigt das entsprechende Erstellungsformular von dem der Benutzer nach erfolgreichem Absenden zur Phabricator-Seite des neu erstellten Sprints weitergeleitet wird.
Bei der Auswertung stellte sich heraus, dass beim letzten Schritt eine Weiterleitung zu Phabricator nicht die beste Lösung war. Bei Sprint-Planning-Meetings hat der Scrum Master meistens jeweils ein Browserfenster für die Sprint-Übersicht und ein Fenster für das Zuweisen der Tasks. Eine Weiterleitung würde ein erneutes Navigieren zu Phragiles Sprint-Übersicht ver-

langen. Ein Lösungsansatz für dieses Problem war die Verwendung einer Bestätigungsseite mit Verweisen zur Sprint-Übersicht, der Sprint-Seite auf Phabricator und einem Link zum Product Backlog in Phabricator.
Zu einem späteren Zeitpunkt, nach der Implementierung der Bestätigungsseite, wurde dieser Ansatz wieder verworfen und stattdessen direkt auf die neu erstellte Sprint-Übersicht weitergeleitet, auf der sich ebenfalls ein Verweis zur Sprint-Seite in Phabricator befindet.

### 4.1.2 Paper Prototype

Paper Prototyping ist eine Methode zum Testen der Interaktionen und verschiedener Anordnungen von Komponenten einer grafischen Oberfläche [Med07]. Dafür werden diese grob auf Papier skizziert, ausgeschnitten und können anschließend beliebig angeordnet werden. Dies bot sich bei Phragile vor allem für die Sprint-Übersicht an.
Hierfür wurden absichtlich dicke Stifte und Marker verwendet, damit der Detailgrad gering bleibt und der Fokus auf der Anordnung liegt. Als Unterlage wurde eine schwarze Pappe gewählt, auf der die ausgeschnittenen Komponenten frei bewegt werden konnten.
Die Auswertung fand wieder mit dem Scrum Master des Wikidata Projekts statt. Die Elemente hatte ich zunächst ähnlich wie bei Scrumbugs - mit dem Burndown-Diagramm und Informationen zu Story Point und einigen Navigationselementen im oberen Teil der Übersicht - angeordnet.

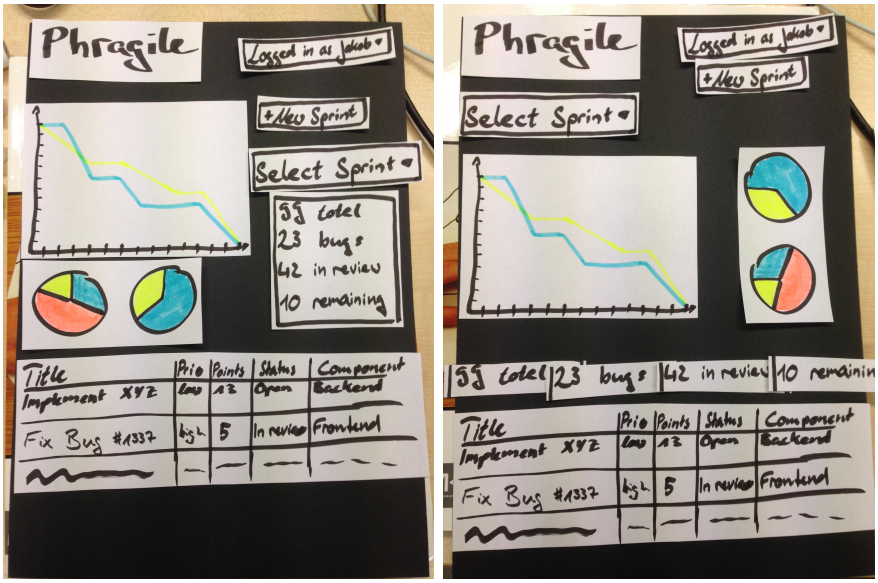
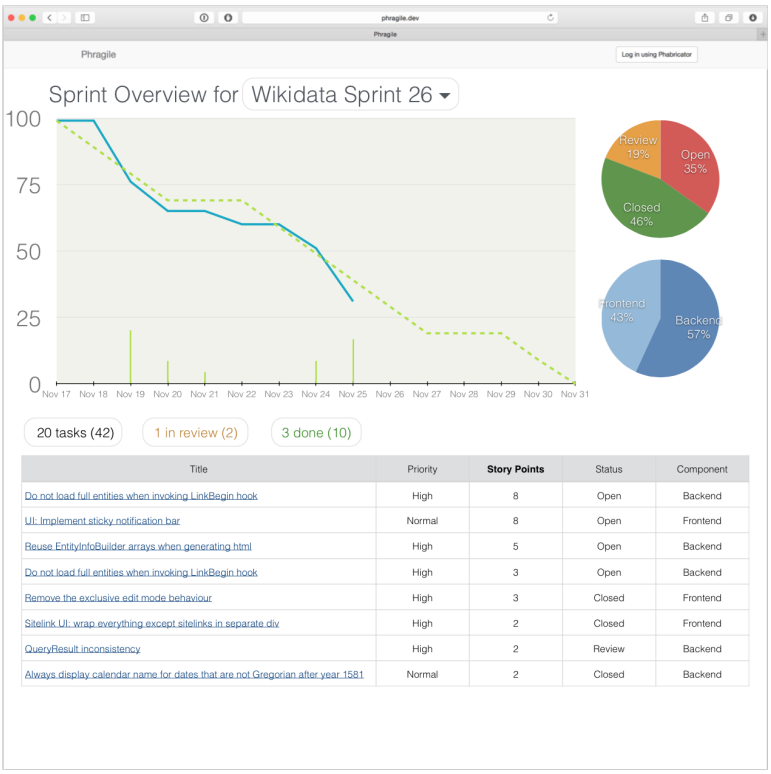



Abbildung 8: Paper Prototypen; Links: Initiale Anordnung; Rechts: Anordnung nach der Auswertung

Es entstand die Idee, dass die Statusinformationen zu Tasks über dem Sprint Backlog als Filter für die Sprint-Backlog-Tabelle funktionieren könnten. Außerdem wurden die Kreisdiagramme neben dem Burndown-Diagramm angeordnet und die Sprint-Auswahl in den Seitentitel eingebunden.

### 4.1.3 Clickdummy

Ein Clickdummy soll dem Benutzer das tatsächliche Verhalten der Applikation an einem Beispiel simulieren [Kel]. Ich habe hierfür einen interaktiven Prototypen in Keynote erstellt. Die Sprint-Übersicht wurde Foliensatz nachgebaut und verschiedene Elemente der Oberfläche mit Verweisen auf eine der insgesamt 15 Folien mit entsprechenden Modifikationen belegt. Beim Klicken auf eine der Tabellenüberschriften des Sprint Backlogs habe ich z. B. auf eine Folie verlinkt, in der ich alle Zeilen dieser Tabelle per Hand entsprechend sortiert habe.
Durch den Clickdummy sollten die Erwartungen des Benutzers und des Entwicklers im Bezug auf die Interaktionsmöglichkeiten der Applikation abgeglichen werden. Um ein möglichst realitätsnahes Szenario hervorzubringen wurde auch die Oberfläche diesmal detaillierter gestaltet.




| Title | Priority | Story Points | Status | Component |
|---|---|---|---|---|
| Do not load full entities when invoking LinkBegin hook | High | 8 | Open | Backend |
| UI: Implement sticky notification bar | Normal | 8 | Open | Frontend |
| Reuse EntityInfoBuilder arrays when generating.html | High | 5 | Open | Backend |
| Do not load full entities when invoking LinkBegin hook | High | 3 | Open | Backend |
| Remove the exclusive edit mode behaviour | High | 3 | Closed | Frontend |
| Sitelink UI: wrap everything except sitelinks in separate div | High | 2 | Closed | Frontend |
| QueryResult inconsistency | High | 2 | Review | Backend |
| Always display calendar name for dates that are not Gregorian after year 1581 | Normal | 2 | Closed | Backend |

Abbildung 9: Folie des Clickdummy mit aktiver Sortierung nach Story Points

Bei der Auswertung gab es bei dieser Methode nur wenig Resonanz, da die Funktionalität zum größten Teil bereits zuvor besprochen wurde und die Interaktionsmöglichkeiten auf den Sprint-Übersichten eher gering und ohne große Überraschungen sind.

## 4.2 Architekturentwürfe und -entscheidungen

Für die Architektur habe ich zunächst in Anwesenheit von Wikimedia Deutschlands Softwarearchitekten die Hauptkomponenten der Software und deren Zusammenspiel am Whiteboard skizziert und erläutert. Die Skizze habe ich nach dem häufig für Webapplikationen verwendeten MVC-Architekturmuster aufgebaut. Dieses dient der Trennung der Benutzeroberfläche von der Domänenlogik, sodass Änderungen an einer der Komponenten isoliert geschehen können und keine Anpassung der jeweils anderen erfordern [Moo].

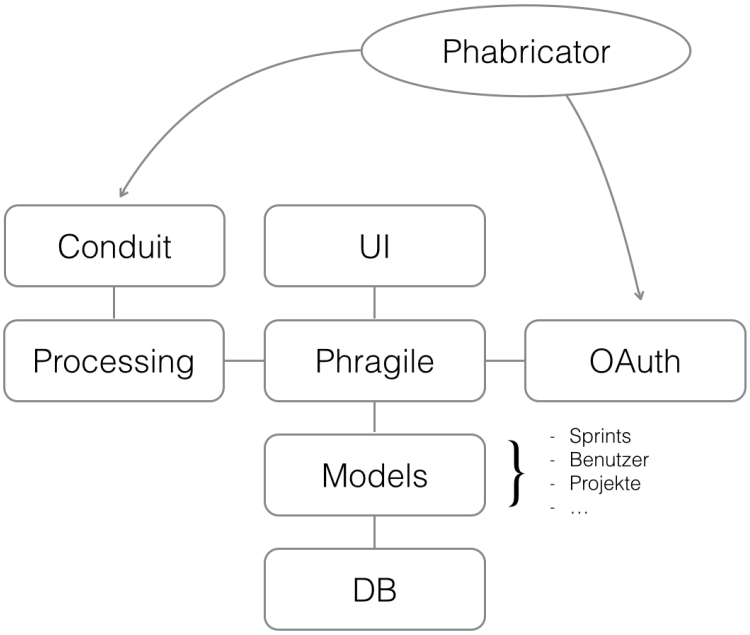



Abbildung 10: Komponenten von Phragile (Whiteboard-Skizze wegen suboptimaler Bildqualität nachgebaut)

*"Phragile"* stellt hierbei den zentralen Punkt der Applikation in Form von Kontroll- und Routing-Logik dar. *"Phabricator"* beschreibt die entfernte Phabricator-Instanz, die mit Phragile über die *"API"*- und *"OAuth"*-Komponenten verbunden ist. An die API angeknüpft befindet sich *"Conduit"*, die klientseitige Schnittstelle, mithilfe derer die Aufrufe an die API abstrahiert werden können. Zur Strukturierung und Aufbereitung möglicher Rückgabewerte der API (z. B. für die Tasks eines Sprints) folgt anschließend eine Komponente, die ich in der Abbildung *"Processing"* genannt habe. Diese Daten

werden schlussendlich über die Kontroll-Logik an die Benutzeroberfläche (*"UI"*) übergeben. Ebenso über Controller mit der grafischen Oberfläche verbunden befinden sich alle Daten, die in Phragiles Datenbank hinterlegt werden. Diese Schicht vor der Datenbank habe ich in der Abbildung nach MVC-Terminologie *"Models"* genannt, wurde allerdings auf die Ambiguität des Begriffs hingewiesen, mit der ich die Persistenzschicht und Logik der Projekt-, Benutzer- und Sprintobjekte beschreiben wollte.

# 5 Funktionsumfang Phragile

Im Folgenden werde ich die Features der Applikation nacheinander erläutern. Diese werden in diesem Kapitel überwiegend aus der Benutzersicht mit nur wenigen technischen Details beschrieben.

## 5.1 Authentifizierung

Die Authentifizierung erfolgt mithilfe von Phabricators eingebautem OAuth-Server-Modul, sodass sich jeder registrierte Phabricator-Benutzer auch bei Phragile anmelden kann. Hierfür findet man in der Navigationsleiste von Phragile einen Button, der zum Authorisierungdialog von Phragile als OAuth-Client für dieses Benutzerkonto führt.

Abbildung 11: Login-Button in der Navigationsleiste

Nach der Bestätigung wird der Benutzer zu Phragile weitergeleitet und ist angemeldet.

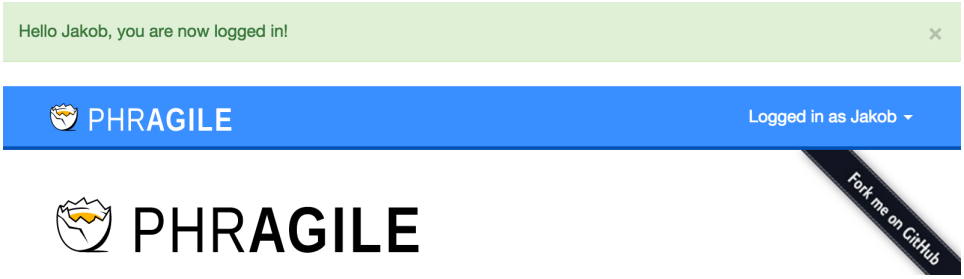



Abbildung 12: Willkommensnachricht nach erfolgreichem Login

## 5.2 Projektliste

Auf der Startseite findet der Benutzer einen “Select a Project”-Button, der eine Liste aller vorhandenen Projekte ausklappt.

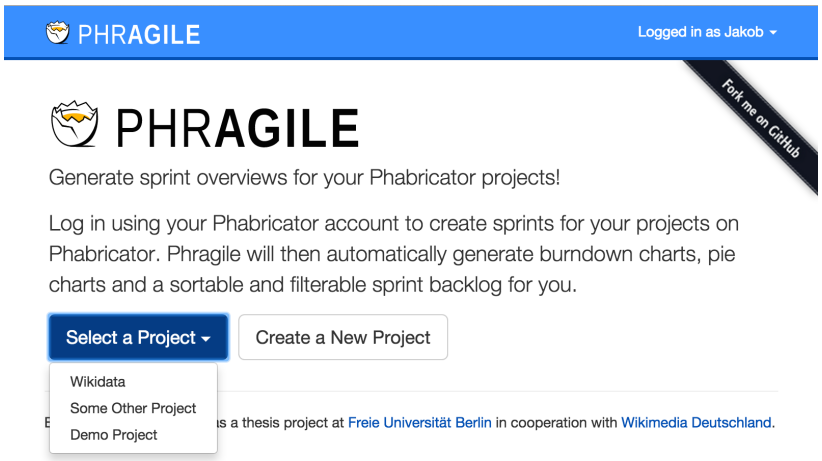



Abbildung 13: Projektliste nach Klick auf “Select a Project”

Klickt man auf eines der Listenelemente, gelangt man zur Projektseite, wo die aktuellste Sprint-Übersicht angezeigt und alle Sprints des Projekts auswählbar sind.

## 5.3 Projekte erstellen

Für jede Phragile-Installation kann festgelegt werden, welche Benutzer der entsprechenden Phabricator-Instanz auf Phragile das Recht haben Projekte zu erstellen. Im eingeloggten Zustand sehen diese Benutzer auf der Startseite neben der Projektauswahl einen weiteren Button zur Erstellung eines neuen Projekts.

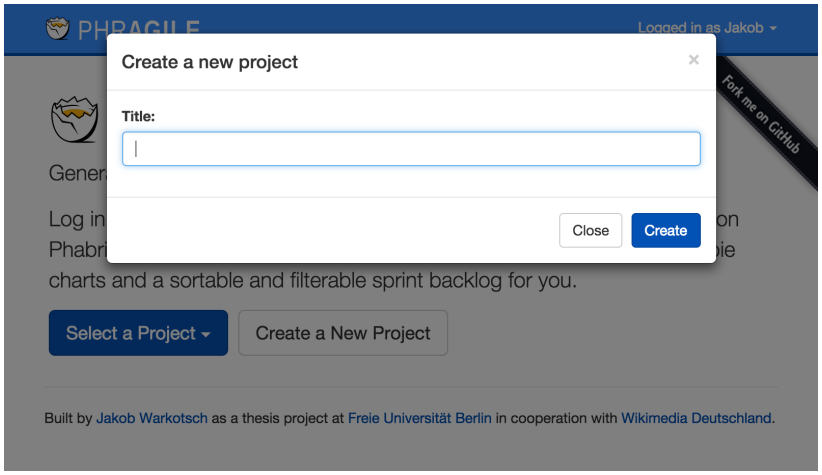



Abbildung 14: Formular zur Projekterstellung nach Klick auf “Create a New Project”

Projekte auf Phragile haben kein entsprechendes Gegenstück auf Phabricator und dienen lediglich Strukturzwecken und der Gruppierung von Sprints.

## 5.4 Sprint-Übersichten

Die Hauptfunktionalität von Phragile besteht in der Generierung von Sprint-Übersichten. Den Hauptteil dieser Übersichten bilden das Burndown-Diagramm, die Sprint-Backlog-Tabelle und das Status-Kreisdiagramm. Des Weiteren findet man auf diesen Seiten eine ausklappbare Liste mit weiteren Sprints desselben Projekts, sowie einen Link zum entsprechenden Phabricator-Projekt.

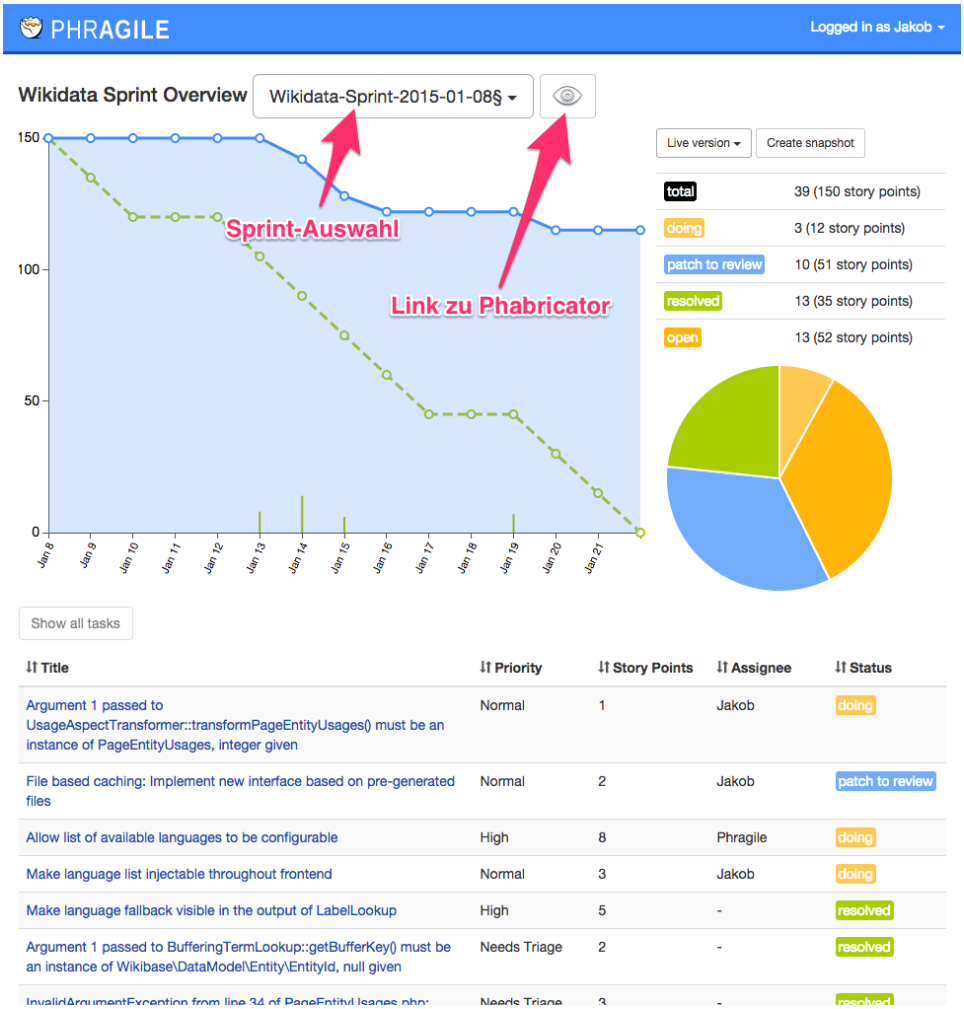



Abbildung 15: Die gesamte Sprint-Übersichtsseite

### 5.4.1 Burndown-Diagramm

Das Burndown-Diagramm visualisiert den Fortschritt des Scrum-Teams für einen Sprint. Auf der X-Achse wird die Dauer des Sprints und auf Y-Achse die Anzahl der verbleibenden Story Points abgebildet. Die Knoten der blauen Linie beschreiben, wie viele Story Points dem Team zu einem Zeitpunkt zum Erreichen des Sprint-Ziels fehlen. Die grüne, gestrichelte Linie entspricht dem Idealverlauf und kann durch den Graphen der folgenden Funktion dargestellt werden:
$f(x) = \text{Gesamtzahl der Story Points} - g(x) * x$
$x$ entspricht hierbei dem x-ten Tag des Sprints und $g(x)$ dem durchschnittlichen Tagesfortschritt, der sich wie folgt berechnet:

$$g(x) = \begin{cases} 0, & \text{wenn } x \text{ Wochenendtag ist} \\ \frac{\text{(Anzahl der Tage im Sprint)} - \text{Wochenendtage}}{\text{Gesamtzahl der Story Points}}, & \text{wenn } x \text{ Wochentag ist} \end{cases}$$

Die grünen Balken, die an der x-Achse des Diagramms ansetzen, visualisieren noch einmal den Fortschritt des Entwicklungsteams am entsprechenden Tag. Fährt man mit der Maus über das Diagramm, werden die der Maus am nächsten Knoten der Linien hervorgehoben und die ideale und tatsächliche Punktzahl an den entsprechenden Knoten wird angezeigt.

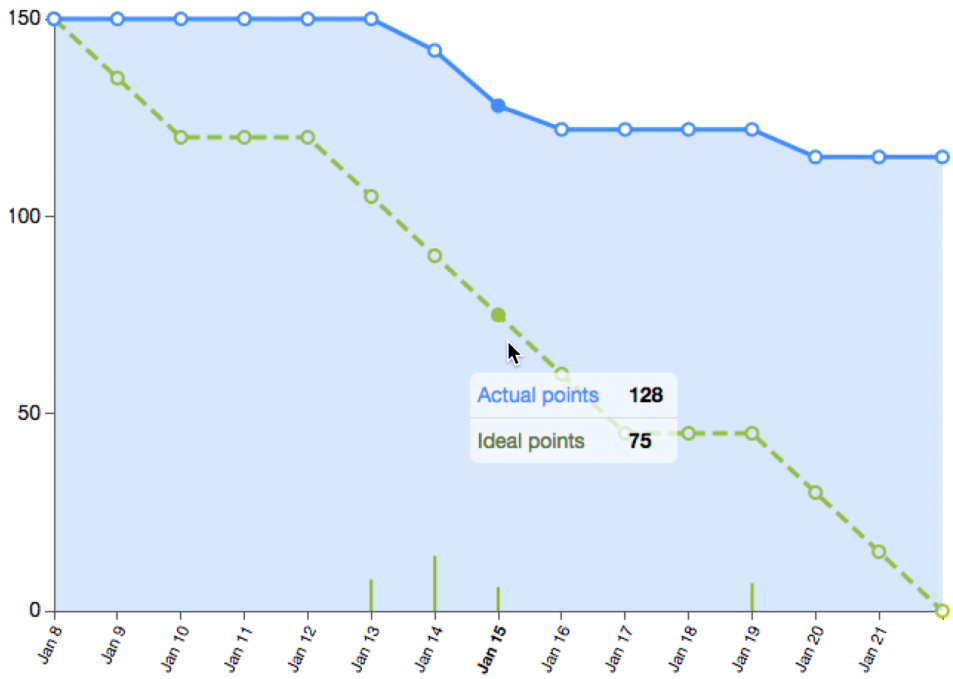



Abbildung 16: Das Burndown-Diagramm beim Mouseover

### 5.4.2 Sprint-Backlog-Tabelle

Diese Tabelle enthält den Titel, die Priorität, die Story Points, den Verantwortlichen, sowie den Status jedes Tasks aus dem Sprint-Backlog des angezeigten Sprints. Per Klick auf eine Spaltenüberschrift werden die Tasks zunächst aufsteigend nach dieser Spalte, bei einem zweiten Klick absteigend sortiert. Des Weiteren gibt es eine Filterfunktion für die Spalten Priorität, Verantwortlicher und Status. Aktiviert wird ein Filter per Klick auf eine der Zellen in der Tabelle. Klickt man beispielsweise auf "open" in der "Status"-Spalte, werden nur noch offene Tasks angezeigt. Zurücksetzbar sind diese Filter durch den dafür vorgesehenen "Show all tasks"-Button außerhalb der Tabelle.

Show all tasks

Filter zurücksetzen

| Title | Priority | Story Points | Assignee | Status |
|---|---|---|---|---|
| File based caching: Decide on file format | Normal | 1 | - | open |
| Give user indication how to influence preferred language selection | Normal | 2 | - | open |
| Apply sticky notifications to new header section | Normal | 2 | - | open |
| Make list of available languages uniform | High | 3 | - | open |
| fix the issue of people not seeing the suggestion in the main search box | High | 3 | - | open |
| File based caching: Script that generates files | Normal | 3 | - | open |
| Have multi-line descriptions | Normal | 3 | - | open |
| Make EntityIdHtmlLinkFormatter show language fallback hint | Normal | 3 | - | open |
| Prepare deployment of usage tracking to Wikidata | Normal | 3 | - | open |
| Mockup for indicating language fallbacks for labels of referenced entities | Normal | 5 | - | open |

Abbildung 17: Das Sprint-Backlog mit aktiven Statusfilter (Open) und Sortierung nach Story Points

### 5.4.3 Status-Kreisdiagramm

Rechts neben dem Burndown-Diagramm befindet sich eine Tabelle, die gleichzeitig als Legende für das darunterliegenden Kreisdiagramms funktioniert. Das Diagramm und die Tabelle zeigen die Verteilung der Status der Tasks des Sprints an. Beim Mouseover über einen der Kreisausschnitte des Diagramms wird der Status für diesen Abschnitt angezeigt. Klicks auf eine Zelle der Tabelle oder einen Kreisausschnitt lösen außerdem die Filterfunktion für den entsprechenden Status im Sprint-Backlog aus.

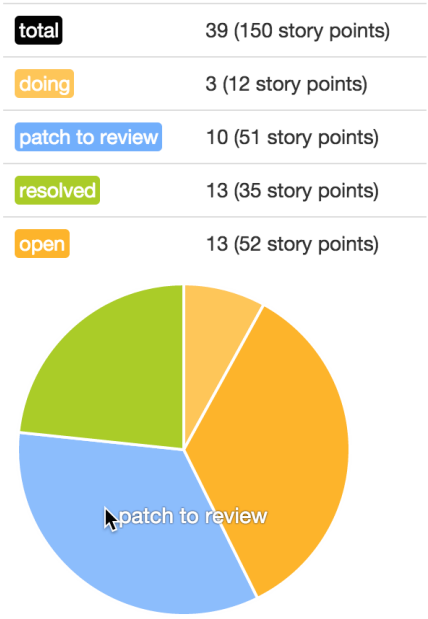



Abbildung 18: Das Statusdiagramm beim Mouseover

## 5.5 Sprints erstellen

Zum Formular zur Erstellung von Sprints gelangt man über einen Klick auf das Dropdown-Menü in der Navigationsleiste, sobald man eingeloggt ist und sich auf einer der projektspezifischen Seiten befindet.

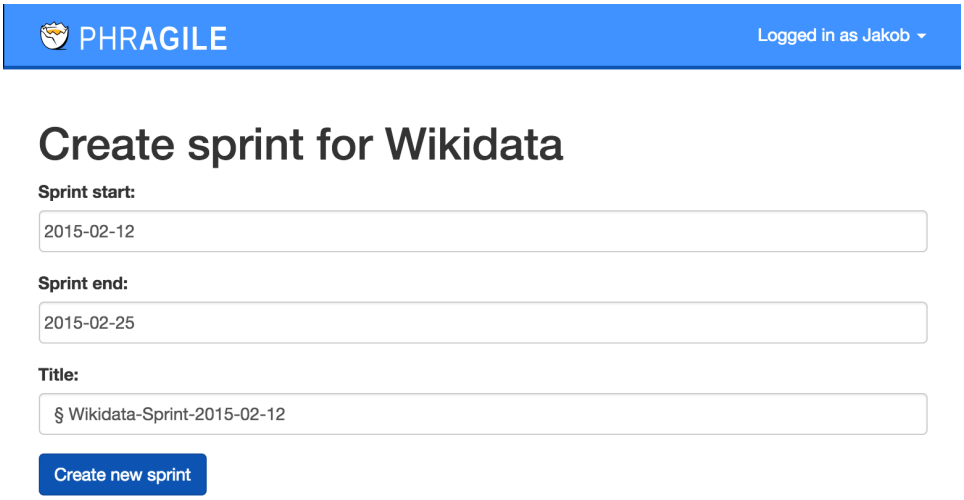



Abbildung 19: Das ausgefüllte Formular zur Erstellung von Sprints

Start- und Enddatum müssen nicht eingetippt werden, sondern erzeugen

beim Fokussieren der Felder per Maus oder Tastatur ein Auswahlfeld für das Datum, in dem man per Klick das gewünschte Datum auswählen kann. Bei der Wahl eines Startdatums wird außerdem automatisch ein Titel der Form "[Projektname]-Sprint-[Startdatum]" für den Sprint generiert und in das "Title"-Feld eingefügt. Dieses bleibt für Sonderfälle weiterhin frei editierbar. Beim Absenden werden die Formulardaten auf das Vorhandensein eines Titels, die korrekte Formatierung von Start- und Enddatum, sowie zeitlichen Reihenfolge (d. h. Start- vor Enddatum) validiert.

## 5.6 Snapshots

Sprint-Snapshots ermöglichen das Archivieren von Sprintdaten zur Retrospektive. Diese Daten werden zur Erstellung des Snapshots einmalig von Phabricator bezogen und anschließend in Phragile gespeichert, sodass keine Aktion auf Phabricator diese verändern kann und eine Einsicht der Daten, wie sie zu genau diesem Zeitpunkt vorlagen, möglich bleibt. Für Sprints, deren Enddatum in der Vergangenheit liegt und für die mindestens ein Snapshot existiert, wird die Sprint-Übersicht dieses Sprints automatisch aus den Daten dieses Snapshots und nicht aus den aktuellen Phabricator-Daten erzeugt. Zur "Live"-Version mit den Daten von Phabricator, sowie zu allen weiteren Snapshots gelangt man über ein Auswahlfeld jeder Sprint-Übersicht. Snapshots lassen sich zu jedem Zeitpunkt von eingeloggten Benutzern sowohl erzeugen, als auch wieder löschen.

## 5.7 Conduit-Zertifikat

Die Einschränkungen des OAuth-Moduls konnten durch das hinterlegen des Conduit-Zertifikat eines Benutzers umgangen werden. Hierzu wird der Benutzer vor der Erstellung des ersten Sprints aufgefordert. Beim Absenden des Formulars führt der Server im Hintergrund eine HTTP-Anfrage zur Überprüfung der Validität des Zertifikats aus und präsentiert dem Benutzer falls nötig eine entsprechende Fehlermeldung.

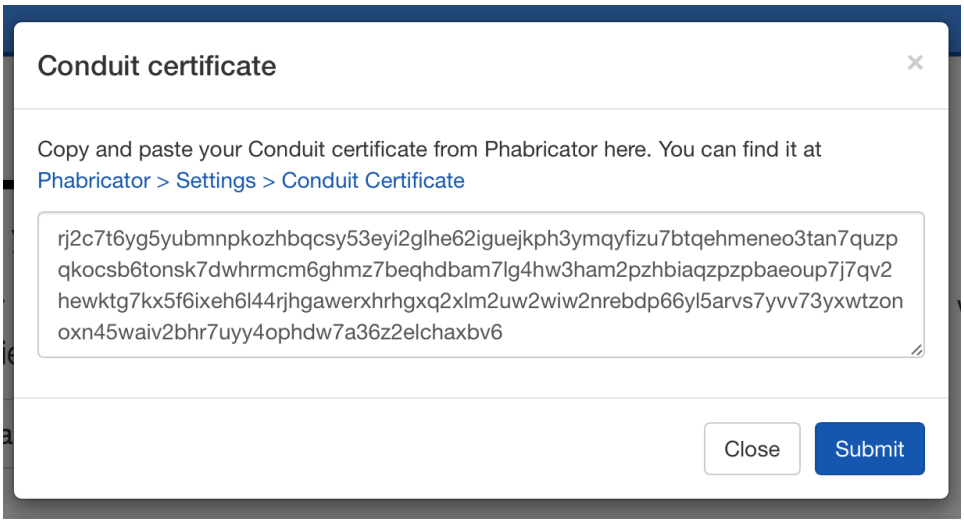



Abbildung 20: Das ausgefüllte Formular zum Hinzufügen eines Conduit-Zertifikats

# 6 Entwicklung

In diesem Kapitel werde ich den Entwicklungsvorgang von Phragile erläutern. Zunächst werde ich die verwendeten Technologien in Front- und Backend beschreiben und anschließend den Ablauf und die Hindernisse schildern.

## 6.1 Technologien

**PHP**
Für die Implementierung des Backends wurde PHP verwendet, damit für die Kommunikation mit Phabricator die PHP-Bibliothek *libphutil* verwendet werden konnte. Diese wurde von Phacility, den Entwicklern von Phabricator, veröffentlicht und enthält viele Hilfsfunktionen und -klassen und die besagte Schnittstelle zu Phabricators Web-API, die unter anderem für das Beziehen der Daten und das Erstellen von Projekten auf Phabricator genutzt wurde. PHP ist außerdem die bevorzugte Sprache der Backend-Entwickler von Wikimedia Deutschland, sodass ich bei Fragen und Code-Reviews stets schnell Hilfe fand und gutes Feedback bekam.

**Laravel**
Laravel[22] ist das in GitHub-Stars [gitb] gemessen bekannteste Web-Framework für PHP. Es vereinfacht viele häufig auftretende Probleme der Webentwicklung wie Authentifizierung, Datenbankabstraktion mithilfe eines ORM, Code-Strukturierung und Entkopplung anhand des MVC-Musters[Moo] und vieles mehr [lar]. Zwei weitere Möglichkeiten waren das Mikro-Framework Silex[23] und das umfassendere Symfony[24]. Laravel wurde wegen der guten Dokumentation und der Einfachheit verwendet. Es stellte außerdem einen guten Kompromiss zwischen der steilen Lernkurve von Symfony und dem eher spartanischen Silex dar.

**Twitter Bootstrap**
Twitter Bootstrap[25] ist ein Framework für HTML-, CSS- und JavaScript-Entwicklung, was das schnelle Erstellen von professionell aussehenden User Interfaces ermöglicht. Es beinhaltet wiederverwendbare und anpassbare Komponenten und Elemente für Buttons, Navigation und Formulare, sowie ein Gittersystem für die Anordnung der Oberfläche[OT].

**LESS**
LESS[26] ist eine Stylesheet-Sprache, die zu CSS kompiliert werden kann. Sie

[22] `http://laravel.com`
[23] `http://silex.sensiolabs.org`
[24] `http://symfony.com`
[25] `http://getbootstrap.com`
[26] `http://lesscss.org`

erweitert CSS um Variablen, Funktionen und ermöglicht das Wiederverwenden von Stilen mithilfe sogenannter Mixins [les].

**D3**
D3 [27] steht für "Data-Driven Documents" und ist eine JavaScript-Bibliothek, mithilfe derer das Burndown- und das Kreisdiagramm entwickelt wurde. Mit D3 lassen sich für das SVG-Format flexible Datenvisualisierungen und vor allem Diagramme erzeugen [Bos].

**List.js**
Für die Sortier- und Filterfunktionalität des Sprint-Backlogs wurde die JavaScript-Bibliothek List.js[28] verwendet. Es vereinfacht Filter- und Sortierfunktionen für HTML-Tabellen und Listenelemente[Str].

## 6.2 Tests

Phragile sollte von Anfang an testgetrieben entwickelt werden. Wegen der Abhängigkeit von externen Komponenten wie Phabricators API und OAuth-Modul und verhältnismäßig wenig Domänenlogik wurde ein BDD-Ansatz gewählt. Hierfür habe ich in der Regel vor der Implementierung einer Funktionalität einen menschenlesbaren Test erstellt, der Zugleich als Dokumentation dieses Features fungierte. Dieser Test wurden mithilfe der PHP-Bibliothek *"Behat"*[29] erzeugt.
Funktionen, die zusätzlich auf der Modulebene viele Daten verarbeiten, wurden außerdem durch *"PHPUnit"*[30]-Tests abgedeckt.

### 6.2.1 Akzeptanztests

Akzeptanztests stellten einen zentralen Punkt der Entwicklung von Phragile dar. Vor der Implementierung eines größeren Features habe ich zunächst eine User Story in Behat-Notation geschrieben und anschließend Wikidatas Scrum Master um Rückmeldung zu dieser gebeten. Behat ist von Rubys Cucumber-Projekt und vor allem der Gherkin-Sprache inspiriert. Eine User Story besteht in Behat aus einer Beschreibung des Features und ggf. mehreren ausführbaren Szenarien. Szenarien sind in drei Blöcke unterteilt, die jeweils einen oder mehrere Schritte enthalten können [beh].

1. *Given* legt Gegebenheiten des Szenarios fest
2. *When* beschreibt die Benutzerinteraktionen

[27] `http://d3js.org`
[28] `http://www.listjs.com`
[29] `http://behat.readthedocs.org/en/v2.5/`
[30] `https://phpunit.de`

3. *Then* beschreibt das gewünschte Resultat des Szenarios

Das Behat-Szenario der Projekterstellung in Phragile z. B. sieht wie folgt aus:

```
Scenario: Project titles have to be unique
  Given I am on the homepage
  And I am logged in
  And the "Wikidata" project exists
  When I fill in "title" with "Wikidata"
  And I press "Create"
  Then I should see "A project with this title already
      exists."
```

Gängige Schrittdefinitionen wie `Given I am on ``x''`, `When I fill in ``x''with ``y''` oder `When I press ``x''` sind bereits in der Behat-Erweiterung "Mink"[31] vordefiniert. Andere, applikationsspezifische Schritte müssen selbst definiert werden. Die Implementierung einer Schrittdefinition sieht z. B. folgendermaßen aus:

```
/**
 * @Given the :title project exists
 */
public function theProjectExists($title)
{
  Project::firstOrCreate([
    'title' => $title,
    'slug' => Str::slug($title)
  ]);
}
```

Mithilfe dieser Tests wurde zum einen der Entwicklungsprozess strukturiert und zum anderen wurden mehrfach Änderungen im Verhalten von Phabricators OAuth-Modul und API frühzeitig erkannt.

### 6.2.2 Unit-Tests

Akzeptanztests sind in Phragiles Fall ein gutes Mittel um auf einer hohen Abstraktionsebene Defekte aufzudecken. Um auf Modulebene Fehler im Code zu erkennen, habe ich *PHPUnit* verwendet, die populärste PHP-Bibliothek für Unit-Testing. Besonders bei der Aufbereitung und Strukturierung von Tasks für das Burndown-Diagramm konnte ich von Modultests profitieren und Phragile gegen Grenzfälle absichern.

## 6.3 Ablauf

Vor Beginn der Implementierung wurde ein Repository auf GitHub für das Projekt angelegt um Code, Dokumentation und Fortschritt der Entwicklung von Anfang an öffentlich zu machen. GitHub ist eine Plattform zur

[31] `http://mink.behat.org/en/latest/`

kollaborativen Softwareentwicklung. Neben der grafischen Oberfläche für das Versionskontrollsystem Git[32], bietet GitHub Code-Hosting, Wikis zur Dokumentation, Code-Reviews und vieles mehr [gita]. Im GitHub-Wiki des Projekts habe ich eine Seite für das Product Backlog und je eine Seite für jedes größere Feature angelegt. Die Wiki-Seite eines Features enthält eine User Story zur groben Beschreibung des Features und außerdem einen Link auf die Behat-Tests dieses Features, die eine detaillierte, schrittweise Beschreibung der Funktionalität enthalten.

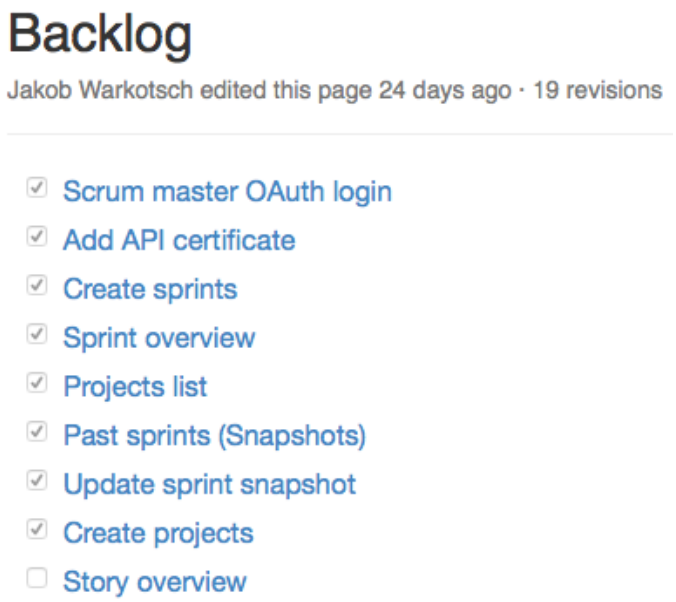



Abbildung 21: Das Product Backlog als Wiki-Seite auf GitHub

Die Entwicklung fand in zweiwöchigen Sprints mit täglichen Stand-Up-Meetings statt. Vor jedem Sprint wurde in einem etwa einstündigen Meeting der vergangene Sprint reflektiert, Resultate präsentiert, Probleme besprochen und der neue Sprint geplant. Bei der Planung wurden Items aus dem Product Backlog in den Sprint aufgenommen, gegebenenfalls in mehrere Tasks unterteilt und bezüglich ihrer Komplexität eingeschätzt. Zur Verfolgung des Fortschritts wurde für jeden Task ein Issue in GitHub angelegt und ein Notizzettel an das Workboard im Büro geklebt. Das Workboard diente als physische Visualisierung des Fortschritts und enthielt zunächst die Spalten "To Do", "Doing" und "Done". Später wurde eine "In Review"-Spalte aufgenommen, da Code Reviews oftmals viel Zeit in Anspruch nahmen und eine übervolle "Doing"-Spalte nicht sehr aussagekräftig ist.

[32] `http://git-scm.com`

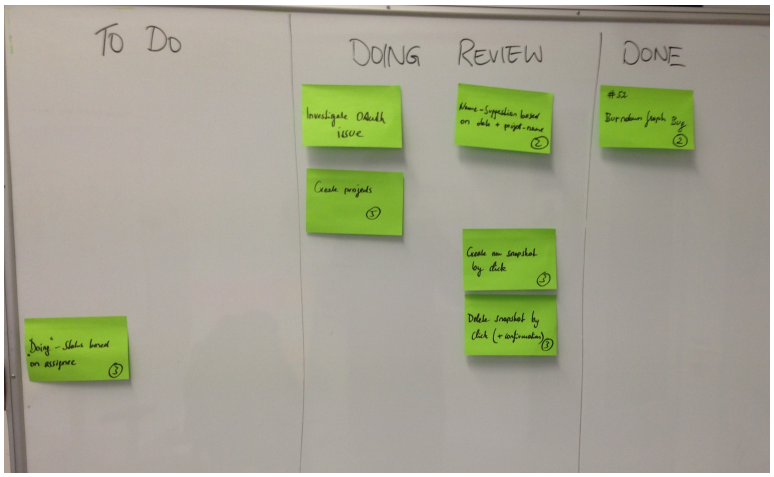



Abbildung 22: Physisches Phragile-Workboard im Büro

**Code-Reviews**
Code Reviews sind eine Methode zur Qualitätssicherung des Programmcodes. Ein Entwickler oder ein Team von Entwicklern veröffentlicht hierbei vor der Verschmelzung mit der eigentlichen Codebasis seine Änderungen am Programmcode um Feedback von anderen Entwicklern einzuholen. In der Regel wird der Code erst dann akzeptiert, wenn alle von den Korrekturlesern erwähnten Probleme gelöst wurden [rev].
Code Reviews waren ein wichtiger Aspekt der Entwicklung von Phragile. Zu Beginn der Arbeit an einem Features habe ich zunächst einen neuen Git-Branch erstellt, dort das Feature bis zum Ende implementiert und anschließend in einem Pull Request auf GitHub um Rückmeldung gebeten. Je nachdem, ob es sich um besonders viel JavaScript oder PHP-Code handelte, bekam ich entweder von Wikimedias JavaScript-Experten oder dem Softwarearchitekten Verbesserungsvorschläge und Feedback zur Sauberkeit und Struktur meiner Implementierung. Erst nachdem alle vom Korrektor angesprochenen Kommentare geklärt waren, galt das Feature als fertig und entsprechende Issues auf GitHub wurden geschlossen und Tasks am Workboard auf “Done” geschoben. Kleinere, triviale Aufgaben wurden gelegentlich auch ohne Review fertiggestellt.

## 6.4 Hindernisse

Bei der Entwicklung einer Softwarelösung verläuft selten alles von Anfang bis Ende wie nach Plan. Im Falle Phragile bin ich dankbarerweise auf keine allzu großen Hürden gestoßen. Im Folgenden werde ich einige meiner Probleme und anschließend den Lösungsweg beschreiben.

### 6.4.1 Phabricator

Phabricator ist eine Plattform, die über 30 verschiedene Komponenten für verschiedene Aufgaben der Projektverwaltung beinhaltet. Änderungen an der Software und dessen Dokumentation werden täglich vorgenommen und befinden sich nach eigener Erfahrung nicht immer auf demselben Stand. Für mich bedeutete das, dass ich in den meisten Fällen der Dokumentation nicht trauen konnte und nur durch Ausprobieren oder das Durchsuchen von Code und Commit-Nachrichten zu einer Lösung gekommen bin.

### 6.4.2 Conduit

Eines des ersten Probleme bei der Arbeit mit Phabricator ist mir bei der Untersuchung der API, *Conduit*, aufgefallen. Einige der Parameter waren nicht gut dokumentiert, andere waren als optional gekennzeichnet und brachten beim Auslassen trotzdem eine Fehlermeldung hervor. Für die Erstellung von Sprints brauchte ich z. B. die Conduit-Methode `project.create`, die einen Parameter für Projekttitel und eine optionale Liste der Projektmitglieder erwartet. Wenn ich nun ein Projekt mit einem Titel, aber ohne Projektmitglieder anlegen möchte, entsteht folgenden Fehlermeldung:

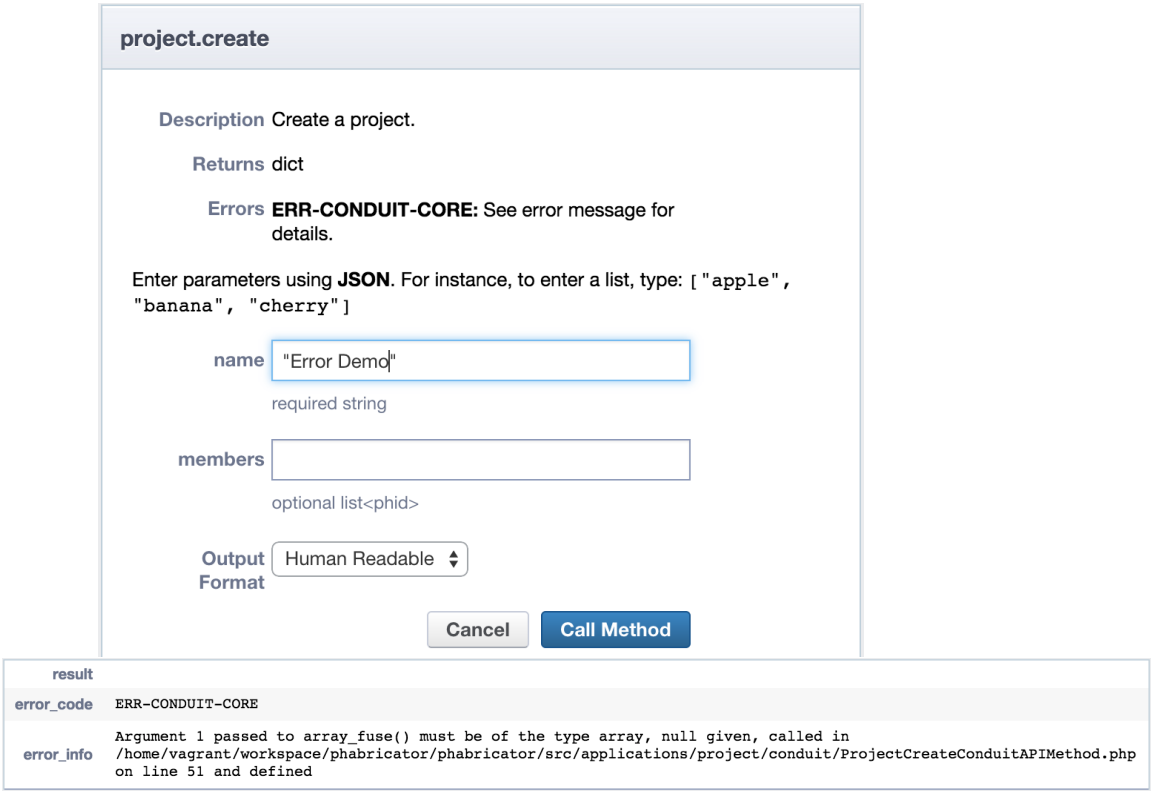



Abbildung 23: Rekonstruktion des Fehlers

Das Absenden des Aufrufs mit einer leeren Liste "[]" funktionierte hingegen, war aber nicht konsistent zum Verhalten der restlichen untersuchten Conduit-Methoden. Da Phabricator zu diesem Zeitpunkt noch Pull Requests auf GitHub entgegennahm beschloss ich zu versuchen, das Problem selbst zu beheben und eine Lösung dort einzureichen. Nach wenigen Minuten konnte ich eine einfache Änderung im Code vornehmen und meinen Pull Request auf GitHub geöffnet.

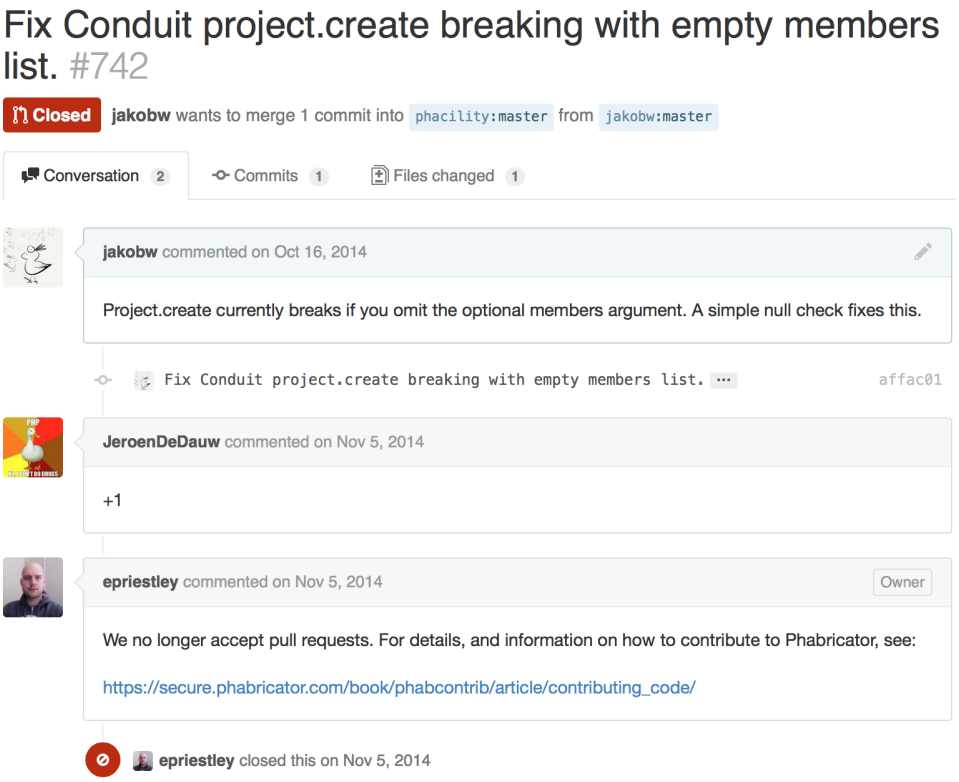



Abbildung 24: Mein Pull Request für Phabricator auf GitHub [phae]

Obwohl in der Woche vor meiner Einreichung noch ein Pull Request[33] akzeptiert wurde, konnte meine Änderung nicht mehr aufgenommen werden, da die Phabricator-Entwickler beschlossen hatten, keine weiteren Pull Requests mehr auf GitHub anzunehmen [phae].
Statt es dort weiter zu versuchen, habe ich weitere Unebenheiten der API für die Entwicklung von Phragile dokumentiert und im Code entsprechend berücksichtigt.

[33] `https://github.com/phacility/phabricator/pull/734`

### 6.4.3 libphutil

*libphutil*, die von Phabricator-Entwicklern erstellte Bibliothek, enthält allgemeine Hilfsfunktionen, sowie das von mir verwendete Interface für Conduit [lib]. Leider werden weder für libphtil, noch für Phabricator PHP-Namensräume verwendet, sodass eine sehr große Menge von Funktionen im globalen Namensraum liegt. Das führte zu einer Namenskollision zweier Funktionen, die auch von dem verwendeten Webframework definiert wurden und somit die offizielle Version von libphutil für mich unbenutzbar machte. Als Lösung dieses Problems habe ich eine öffentliche Kopie[34] von libphutil erstellt und die zwei problematischen Methoden umbenannt. Solange libphutil keine Namensräume verwendet oder die Conduit-Schnittstelle als eigenes Modul bereitgestellt wird, wird Phragile meine Version der Bibliothek beziehen, die ich dank Git ohne Schwierigkeiten aktuell halten kann.

### 6.4.4 OAuth

Die Arbeit mit dem sich noch im "Prototype"-Status befindendem OAuth-Modul von Phabricator war - wie zu erwarten - nicht durchgehend problemfrei. Bereits zu Beginn der Implementierung der Login-Funktionalität musste ich feststellen, dass der OAuth-Dienst sich nicht überall der Dokumentation entsprechend verhielt. In der Dokumentation heißt es, dass bei erfolgreicher Durchführung des OAuth-Handshakes, der Benutzer am Ende mit einem Access Token zurück zur Dritt-Applikation (d. h. in meinem Fall Phragile) *weitergeleitet* wird. "If done correctly, the OAuth Server will *redirect* to the pertinent redirect_uri with an access token." Tatsächlich gibt es aber keine Weiterleitung, sondern eine Antwort des Servers mit einem JSON-String in dem sich der Access Token befindet. Ich habe zunächst den Fehler in meiner Benutzung der Phabricator-Funktionalität gesucht, konnte aber selbst im Code der OAuth-Komponente von Phabricator nur die JSON-Antwort und keine Weiterleitung finden.

Die Lösung des Problems bestand darin, dass ich, statt den Benutzer selbst auf Phabricator weiterzuleiten, einen HTTP-Request auf die entsprechende Phabricator-URL mit dem Authorization Code des Benutzers mache und den Access Token aus der JSON-Antwort lese. Außerdem habe ich hierzu ein Ticket in Phabricator geöffnet, was allerdings auch direkt wieder geschlossen wurde, da Fehlermeldungen zu "Prototype"-Komponenten unerwünscht sind[phaf].

[34] `https://github.com/jakobw/libphutil`

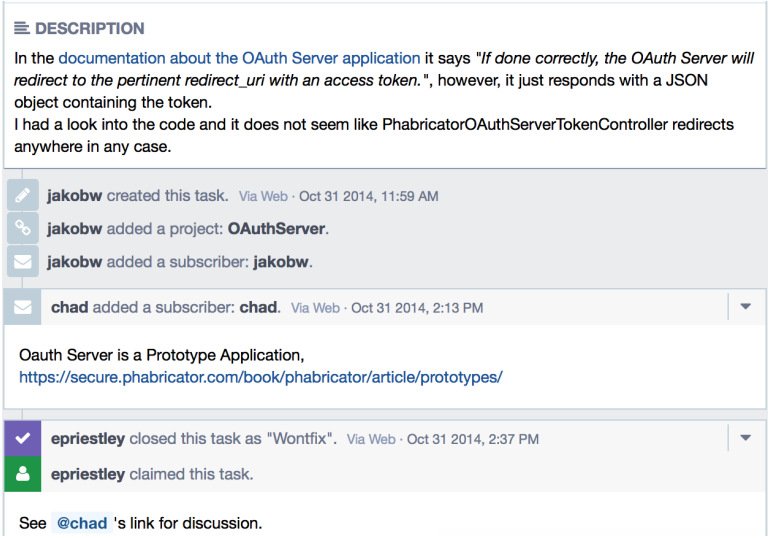



Abbildung 25: Mein Ticket zu Phabricators OAuth-Dienst

Das zweite Problem mit OAuth entstand kurz vor Abschluss der Entwicklung der ersten Version von Phragile. Aufgedeckt wurde das Problem durch das Fehlschlagen aller Behat-Tests, die einen Login voraussetzten, nach einem Upgrade auf die aktuellste Phabricator-Version. Die neue Version brachte eine Änderung mit sich, die das Verhalten der Verwendung des Access Tokens wie folgt änderte:

1. Der HTTP-Parameter wurde von “access_token” zu “api.access_token” geändert.

2. Das Conduit-Interface aus *libphutil* konnte hierfür nicht mehr verwendet werden. Das Interface setzt zum Ausführen beliebiger Conduit-Methoden eine Verbindung mit einem Phabricator-Konto voraus. Zu diesem Zweck wurde für Phragile ein Bot-Benutzer (d. h. Benutzer ohne Passwort) für nicht-benutzerspezifische Aufgaben wie das Abfragen aller Tasks eines Sprints verwendet. Nach der Änderung gab die Conduit-Methode `user.whoami` statt der Daten des Benutzers, die Benutzerdaten des verbundenen Bot-Benutzers zurück und ignorierte den Access Token. [phaa]

Keine dieser Änderungen wurde in der Dokumentation des OAuth-Moduls erfasst und konnte nur durch Suchen entsprechender Commits und Code auf GitHub erkannt werden. Um die Login-Funktionalität wiederherzustellen habe ich die Verwendung der Conduit-Schnittstelle zu einem normalen HTTP-Request geändert und den Query-Parameter entsprechend geändert.

# 7 Fazit

In diesem abschließenden Abschnitt werde ich das Ergebnis dieser Arbeit zusammenfassen und einen Ausblick auf mögliche Verbesserungen und Erweiterungen geben.

## 7.1 Ausblick

Phragile kann in vielerlei Hinsicht erweitert und verbessert werden. Neben kleineren Änderungen, die zum Teil in den GitHub Issues [35] dokumentiert sind, möchte ich im Folgenden drei Möglichkeiten für neue Funktionalitäten präsentieren.

**User Stories**
Eine naheliegende Option ist die Implementierung der in Tabelle2 erwähnten Unterstützung für User Stories, die dem Product Owner eine Übersicht über den Status eines Features auf Projektebene gibt, ohne dessen Teilaufgaben einzeln zu betrachten. Das setzt voraus, dass zuvor alle Teilaufgaben dieser Funktionalität festgelegt und deren Umfang abgeschätzt wurde.
Da Phabricator über kein entsprechendes Konzept verfügt, müsste zunächst untersucht werden, ob und wie User Stories abgebildet werden können. Ein möglicher Ansatz wäre die Verwendung von Tasks als User Stories und Subtasks als deren Teilaufgaben.

**Workboards**
Neben der Zuweisung verschiedener Status, unterstützt Phabricator die Nutzung von Workboards [man], in denen Aufgaben in benutzerdefinierten Spalten entsprechend ihres Arbeitsstatus verschoben werden können. Bisher bezieht Phragile den Status nur über das Statusfeld der jeweiligen Aufgabe. Das Scrum Team von Wikidata möchte in Zukunft mehr Gebrauch von Phabricators Workboards machen um somit die Benutzung von Kanban mit Phabricator intuitiver zu machen und die physische Tafel besser abzubilden. Zu untersuchen bleibt, wie die Abfrage der Workboard-Daten funktioniert und ob diese Lösung mithilfe Phabricators Webschnittstelle realisierbar ist.

**Integration in Phabricator**
Obwohl Phragile unter anderem wegen mangelnder Unterstützung integrierter Applikationen als externe Applikation entwickelt wurde, bin ich der Meinung, dass die Einbettung von Phragile in Phabricator, sobald eine entsprechende Schnittstelle stabil ist, langfristig eine bessere Lösung darstellt. Eine Integration würde zum einen die Benutzung intuitiver machen, indem Sprint-Übersichten direkt als Unterseiten der Phabricator-Sprints aufrufbar

[35] `https://github.com/wmde/phragile/issues`

wären. Zum anderen wären die vielen Notlösungen, wie z. B. die Erstellung der Sprints aus Phragile, überflüssig. Die Nutzung einer externen Applikation scheint – auch wenn das Scrum Team von Wikidata es durch die Nutzung von Scrumbugs gewöhnt war – nicht im Sinne von Phabricators Philosophie einer zentralen Plattform[phac] für Softwareentwicklung zu sein.

## 7.2 Resultat

Das Resultat meiner Arbeit ist eine funktionierende Applikation zur Generierung von Sprint-Übersichten aus Phabricator-Projektdaten, die alle vor Beginn der Implementierung festgelegten Anforderungen erfüllt. Eine Testinstallation der Anwendung wurde bereits von Wikidatas Scrum Team verwendet und die Inbetriebnahme erfolgt, sobald Wikimedias Phabricator-Instanz entsprechend konfiguriert wurde. Der Programmcode, sowie Installationshinweise und die Dokumentation des Funktionsumfangs wurden auf GitHub[36] unter der GNU General Public License Version 2[37] veröffentlicht, sodass die Software auch von Drittbenutzern verwendet und erweitert werden kann.

[36] `https://github.com/wmde/phragile`
[37] `http://www.gnu.org/licenses/gpl-2.0.html`

# Literatur

[AMKM11] Gayane Azizyan, Miganoush Katrin Magarian, and Mira Kajko-Matsson. Survey of agile tool usage and needs. In *Agile Conference (AGILE), 2011*, pages 29–38. IEEE, 2011.

[beh] Behat documentation. http://behat.readthedocs.org/en/v2.5/. Abruf: 15.02.2015.

[Bos] Mike Bostock. D3.js - data-drive documents. http://d3js.org. Abruf: 16.02.2015.

[GCMB11] Saul Greenberg, Sheelagh Carpendale, Nicolai Marquardt, and Bill Buxton. *Sketching user experiences: The workbook*. Elsevier, 2011.

[gita] Github features. https://github.com/features. Abruf: 31.01.2015.

[gitb] Web application frameworks - github. https://github.com/showcases/web-application-frameworks?s=stars. Abruf: 16.02.2015.

[Kel] Amir Kella. Creating interactive prototypes with keynote. http://keynotopia.com/guides/. Abruf: 03.02.2015.

[lar] Laravel documentation. http://laravel.com/docs/5.0. Abruf: 16.02.2015.

[les] Less documenation. http://lesscss.org. Abruf: 15.02.2015.

[lib] libphutil technical documentation. https://secure.phabricator.com/book/libphutil/. Abruf: 18.02.2015.

[man] Phabricator - maniphest. http://phabricator.org/applications/maniphest/. Abruf: 22.02.2015.

[Med07] Shawn Medero. Paper prototyping. http://alistapart.com/article/paperprototyping, 07 2007. Abruf: 03.02.2015.

[Moo] Jeff Moore. Model view controller. http://www.phpwact.org/pattern/model_view_controller. Abruf: 02.03.2015.

[oaua] Oauth community site. http://oauth.net/. Abruf: 21.02.2015.

[oaub] Using the phabricator oauth server. https://secure.phabricator.com/book/phabcontrib/article/using_oauthserver/#scopes. Abruf: 23.02.2015.

[OT] Mark Otto and Jacob Thornton. Twitter bootstrap documentation. http://getbootstrap.com/components/. Abruf: 16.02.2015.

[PAU14] GUILLAUME PAUMIER. On our way to phabricator. http://blog.wikimedia.org/2014/06/10/on-our-way-to-phabricator/, June 2014. Abruf: 27.01.2015.

[phaa] Accept conduit tokens as an authentication mechanism. https://github.com/phacility/phabricator/commit/0507626f01e52d27f50e299f2f7eac0fd9a14 Abruf: 25.02.2015.

[phab] How do i write my own phabricator applications? https://secure.phabricator.com/T5447. Abruf: 20.02.2015.

[phac] Phabricator - applications. http://phabricator.org/applications/. Abruf: 21.02.2015.

[phad] Phabricator custom fields. https://secure.phabricator.com/book/phabricator/article/custom_fields/. Abruf: 21.02.2015.

[phae] Phabricator pull request. https://github.com/phacility/phabricator/pull/742. Abruf: 25.02.2015.

[phaf] User guide: Prototype applications. https://secure.phabricator.com/book/phabricator/article/prototypes/. Abruf: 18.02.2015.

[phag] What is phabricator. https://secure.phabricator.com/book/phabricator/article/introduction/#what-is-phabricator. Abruf: 01.03.2015.

[rev] Codereview. https://sdqweb.ipd.kit.edu/wiki/Codereview. Abruf: 31.01.2015.

[scr] Scrumbugs help. https://scrumbu.gs/help/. Abruf: 01.03.2015.

[Sofa] Mountain Goat Software. Sprint backlog and the scrum sprint. http://www.mountaingoatsoftware.com/agile/scrum/sprint-backlog. Abruf: 10.02.2015.

[Sofb] Mountain Goat Software. User stories. http://www.mountaingoatsoftware.com/agile/user-stories. Abruf: 10.02.2015.

[SS11] Ken Schwaber and Jeff Sutherland. The scrum guide. *Scrum Alliance*, 2011.

[Str] Jonny Strömberg. List.js documentation. http://www.listjs.com. Abruf: 15.02.2015.

[wika] 2014-10-14 wikidata developers. http://sb.wmflabs.org/t/wikidata-developers/2014-10-14/. Abruf: 04.03.2015.

[wikb] Verein - wikimedia deutschland. https://www.wikimedia.de/wiki/%C3%9Cber_uns. Abruf: 04.03.2015.