31. Oktober 2023

# Software mit agilen Methoden entwickeln

Patrick Côté,
Carl Strömstedt,
Melina Baumann,
Sarah Kalbermatter,
Francesca Scarfi

Baum
Bros

# Inhalt

# Projektübersicht

In diesem Projekt haben wir die Umsetzung unserer Webanwendung «Baumbro» mithilfe der agilen Projektmanagementmethode *Scrum* durchgeführt.

Das Ziel dieses Projekts war es, mithilfe von Open Data der Stadt Zürich und Geodaten den Benutzern die Möglichkeit zu bieten, Bäume in ihrer Umgebung zu lokalisieren und detaillierte Informationen zu diesen Bäumen abzurufen. Die Anwendung ermöglicht es den Benutzern, sich auf der «Baumbro»-Website zu registrieren, sich anzumelden, ihre Lieblingsbäume zu verfolgen und eigene Bilder von Bäumen hochzuladen.

«Baumbro» zielt darauf ab, die Verbindung zwischen der städtischen Umwelt und den Bewohnern von Zürich zu stärken, indem es Informationen über die Bäume in der Stadt zugänglicher und interaktiver macht. In dieser Dokumentation werden wir die agilen Praktiken und Rituale, die während des Projekts verwendet wurden, im Detail erläutern und die Herausforderungen und Erfolge des Teams beschreiben.

# Scrum Team

Unser Scrum-Team für das Projekt «Baumbro» setzte sich aus den folgenden Teammitglieder zusammen:

**Product Owner (PO):**

- Patrick Côté

**Entwicklungsteam (DEV):**

- Carl Strömstedt
- Melina Baumann
- Sarah Kalbermatter

**Scrum Master (SM):**

- Francesca Scarfi

# Scrum Rituale

Im Folgenden werden allgemeine Eckpunkte von *Scrum* skizziert und dokumentiert, wie die regelmässigen Meetings bei uns abliefen, ergänzt mit einer Reflexion oder allfälligen Verbesserungsvorschlägen für die Zukunft.

## Artefakte

Zu den wichtigen Artefakten gehören der *Product Backlog*, der *Sprint Backlog* und das *Inkrement*. Im *Product Backlog* halten wir alle geplanten Funktionen und Aufgaben für unser Projekt fest. Der *Sprint Backlog* enthält die spezifischen Aufgaben, die in einem Sprint umgesetzt werden, und das *Inkrement* zeigt den aktuellen Stand des Projekts. Diese Artefakte ermöglichen es uns, klare Prioritäten zu setzen, den Fortschritt zu überwachen und sicherzustellen, dass wir auf dem richtigen Weg sind. Sie dienen auch als Grundlage für unsere *Sprint*-Planung und tragen dazu bei, dass unser Team fokussiert und effizient arbeitet.

Der Product Backlog wird im Kapitel «*Scrum Tools*» noch ergänzt.

### *Sprint Planning*

Unsere Sprintplanungen wurden regelmässig am Anfang jedes *Sprints* durchgeführt. In diesen Meetings definierten wir die Ziele für den bevorstehenden *Sprint* und legten fest, welche Aufgaben in den *Sprint Backlog* aufgenommen werden sollten. Die *Story Points* zur Bewertung der Aufgaben haben wir mithilfe eines Planning Poker-Spiels vergeben. Diese Methode ermöglichte es dem Entwicklungsteam, eine gemeinsame Einschätzung des Aufwands vorzunehmen, indem die Teammitglieder ihre Schätzungen diskutierten und ihre unterschiedlichen Perspektiven einbrachten.

### *Dailys*

Unser *Stand Up*-Meeting fand an jedem Schultag statt und wurde mithilfe eines Dashboards in JIRA durchgeführt. Dieses Dashboard bot eine transparente Übersicht darüber, an welchen Aufgaben jedes Teammitglied arbeitete. Die Dailys waren auf 15 min timeboxed: Wir stellten fest, dass es uns nicht schwerfiel, diese Vorgabe einzuhalten, denn klare und einfache Themen konnten kurz und prägnant zusammengefasst werden, und Tasks mit spezifischen Herausforderungen betreffen sowieso nicht das gesamte Team und sollten im Anschluss bilateral angegangen werden.

## Retrospektive

Unser *Retro*-Meeting wurde in Form eines Retroschiffs durchgeführt, und wir begannen mit den positiven Aspekten, die uns halfen, eine positive und konstruktive Atmosphäre zu schaffen.

Eine wichtige Erkenntnis, die wir für das nächste Mal mitnehmen, ist, dass wir das Feedback anonym sammeln und dann gemeinsam besprechen möchten. Diese Methode ermöglicht es uns, ehrliches Feedback ohne Angst vor möglichen Konsequenzen zu erhalten und kritischere Einblicke in die Stärken und Schwächen des Teams und des Prozesses zu gewinnen.

## Review

Am Ende eines jeden Sprints organisierten wir ein *Sprint Review*-Meeting, bei dem unser Product Owner, der die Stakeholder des Projekts vertrat, Feedback gab. Dies half sicherzustellen, dass das Produkt den Erwartungen und Anforderungen der Stakeholder entsprach. Das Entwicklungsteam präsentierte neue Funktionen und Meilensteine, und das erhaltene Feedback trug zur kontinuierlichen Verbesserung des Produkts bei, um den Bedürfnissen unserer Stakeholder gerecht zu werden.

## Refinement

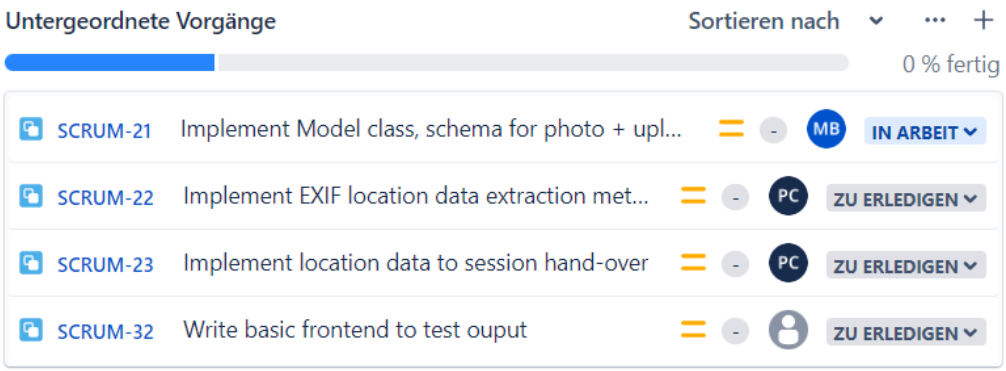
Wir führten regelmässige *Backlog Refinement*-Sitzungen durch, um den Product Backlog zu überarbeiten und sicherzustellen, dass die Prioritäten und Anforderungen klar definiert waren.

# *Scrum* Tooling

Im folgenden Kapitel wird auf die verwendeten *Scrum*-Tools genauer eingegangen.

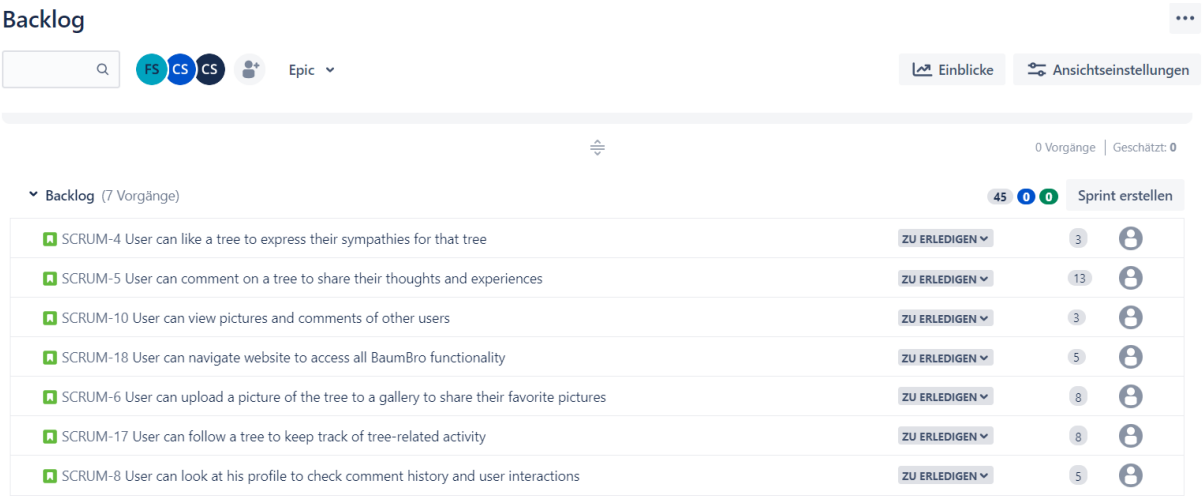
## JIRA

In unserem Projekt verwendeten wir die Projektmanagement-Software JIRA als zentrales Tool, um unsere Aufgaben zu verfolgen und die Arbeit effizient zu organisieren. Wir nutzten JIRA, um *User Stories* zu erstellen, die die funktionalen Anforderungen des Projekts abbildeten. Diese *User Stories* wurden dann in kleinere Aufgaben, sogenannte *Subtasks*, unterteilt, um die Arbeit detaillierter zu planen und den Fortschritt besser zu überwachen.

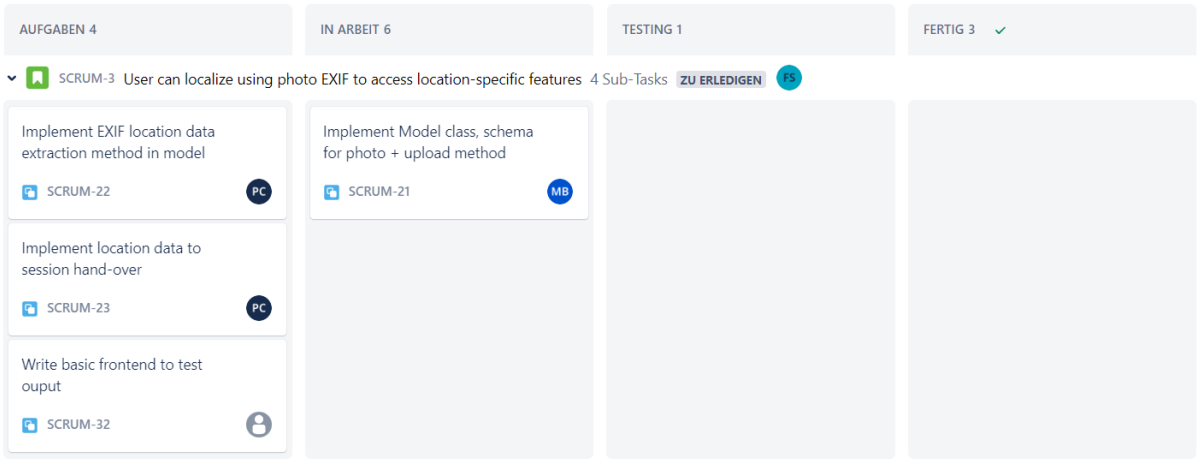



Nebst *User Stories* und *Subtasks* erstellten wir auch einen *Product Backlog* in JIRA, in dem wir alle geplanten Funktionen und Aufgaben für das Projekt verwalten und priorisieren konnten.

Er half uns, die höchstpriorisierten Aufgaben in den *Sprint Backlog* zu ziehen und daraus resultierend einen erfolgreichen *Sprint* durchzuführen, mit dessen *Inkrement* unsere Stakeholder zufriedenzustellen waren.



Zusätzlich verwendeten wir ein Dashboard in JIRA, das speziell für unser Projekt erstellt wurde. Dieses Dashboard bot eine übersichtliche Darstellung von Projektmetriken, Aufgabenstatus und Fortschrittsinformationen. Es ermöglichte dem Team, in Echtzeit zu sehen, an welchen Aufgaben gearbeitet wurde, den Fortschritt zu überwachen und Hindernisse zu identifizieren.




## Definition of Done

Die *Definition of Done* (DOD) ist ein entscheidendes Element in unserer agilen Entwicklungspraxis. Sie stellt sicher, dass jede abgeschlossene *User Story* unseren Qualitätsstandards entspricht, bevor sie als «fertig» betrachtet wird. Die Festlegung klarer und strikter DOD-Kriterien ist für uns von zentraler Bedeutung, da sie Transparenz und Klarheit über den Status und die Qualität unserer Arbeit bietet. Nachfolgend werden die spezifischen Punkte unserer DOD erläutert und warum wir diese Kriterien gewählt haben.

**DOD**

| | |
|---|---|
| Code reviewed (four-eyes principle) | |
| Acceptance criteria met | |
| Non-functional tests passed | |
| Merge Request in Github is accepted | |

**Code Review (Four-Eyes Principle):** Die Auswahl des Code-Reviews als ersten DOD-Punkt basiert auf unserem Engagement für qualitativ hochwertige Software. Der Einsatz des «Four-Eyes Principle» gewährleistet, dass alle von uns entwickelten Codes nicht nur von einem Entwickler, der ihn geschrieben hat, überprüft wird, sondern auch von einem anderen Teammitglied. Dieser Prozess fördert die Zusammenarbeit und ermöglicht es uns, potenzielle Fehler oder Verbesserungen frühzeitig zu identifizieren und zu beheben.

**Acceptance Criteria Met:** Die Erfüllung der Akzeptanzkriterien ist ein wesentlicher Schritt, um sicherzustellen, dass die entwickelten Features und Funktionen den Kundenanforderungen entsprechen. Wir haben dieses Kriterium aufgenommen, um sicherzustellen, dass unsere Arbeit nicht nur technisch korrekt ist, sondern auch den eigentlichen Zweck erfüllt, den die Kunden erwarten.

**Non-Functional Tests Passed:** Nicht nur funktionale Anforderungen sind entscheidend, sondern auch nicht-funktionale Anforderungen wie Leistung, Sicherheit und Zuverlässigkeit. Diese DOD-Regelung stellt sicher, dass wir auch in diesen wichtigen Bereichen den höchsten Qualitätsstandards entsprechen.

**Merge Request in Github is Accepted:** Die Annahme des Merge Requests in GitHub ist ein abschliessender Schritt, um sicherzustellen, dass der entwickelte Code in die Hauptcodebasis integriert wird. Dadurch wird gewährleistet, dass unsere Arbeit nicht nur auf einem Entwicklungszweig isoliert bleibt, sondern erfolgreich in die Produktionsumgebung überführt wird.

# Versionsverwaltung

In unserer agilen Entwicklungsumgebung ist die Versionierung von Code von entscheidender Bedeutung. Für die Versionierung unserer Projekte haben wir uns bewusst für GitHub entschieden, und in diesem Kapitel möchten wir erläutern, warum diese Plattform für uns die passende Wahl ist.

## Github

GitHub ist eine weit verbreitete und hoch angesehene Versionsverwaltungsplattform, die sich durch ihren Funktionsumfang und ihre Benutzerfreundlichkeit auszeichnet. Hier sind einige der Gründe, warum wir uns für GitHub entschieden haben:

GitHub bietet umfangreiche Kollaborationswerkzeuge, darunter Diskussionsmöglichkeiten, Kommentarfunktionen, Aufgabenverfolgung und die Option, Code- und Dokumentationsänderungen unkompliziert zu überprüfen und zu genehmigen. Dies fördert Zusammenarbeit und Kommunikation im Team.

Die Plattform verfügt zudem über starke Sicherheitsfunktionen, die unseren hohen Sicherheitsstandards gerecht werden. Dazu gehören Zugriffskontrollen, Schwachstellenmanagement und die Möglichkeit, Sicherheitsscans direkt in den Entwicklungsprozess zu integrieren.

Nicht zuletzt ermöglicht GitHub eine nahtlose Integration mit JIRA, unserem bevorzugten Projektmanagement- und Aufgabenverfolgungstool. Diese Integration erleichtert die Verknüpfung von Codeänderungen mit den entsprechenden *User Stories* oder *Subtasks* in JIRA und fördert somit eine effiziente und transparente Arbeitsweise.

Um sicherzustellen, dass wir eine klare Verbindung zwischen unseren Codeänderungen und den zugehörigen Aufgaben in JIRA haben, erstellten wir pro Jira-Nummer einen Branch, welcher nach der Vieraugenkontrolle gemerget wurde. Dies bietet uns folgende Vorteile:

- **Transparenz**: Die Verwendung derselben Nummern in GitHub wie in JIRA ermöglicht es jedem im Teammitglied, schnell zu erkennen, welcher Code zu welcher Aufgabe gehört. Dies fördert die Transparenz und erleichtert die Kommunikation.
- **Nachvollziehbarkeit**: Wir können den Verlauf und den Fortschritt einer Aufgabe in JIRA einfach mit den zugehörigen Codeänderungen in GitHub abgleichen. Dies erleichtert die Nachvollziehbarkeit und Qualitätskontrolle.
- **Effizienz**: Die konsistente Verwendung von Nummern in beiden Plattformen erleichtert die Zusammenarbeit zwischen Entwicklern, Testern und anderen Teammitgliedern. Jeder weiss genau, welchen Teil des Projekts sie überprüfen oder testen müssen.

## Softwarearchitektur

Unsere Softwareanwendung basiert auf dem Python-Flask-Framework und implementiert eine einfache Webanwendung. Die Architektur unserer Anwendung ist nach dem MVC (Model-View-Controller) -Muster strukturiert. Im Detail bedeutet dies:

Model: Dieser Teil unserer Anwendung enthält die Datenbankmodelle und die Datenbankverwaltung. Wir verwenden eine relationale Datenbank, um Benutzerdaten zu speichern.

View: Die Ansichten in unserer Anwendung sind in HTML-Vorlagen implementiert, die mithilfe des Flask-Templating-Mechanismus gerendert werden. Die Benutzeroberfläche ist einfach und benutzerfreundlich gestaltet.

Controller: Die Steuerung und Logik unserer Anwendung sind in den Python-Funktionen in diesem Skript implementiert. Die Routen definieren, wie HTTP-Anfragen verarbeitet werden, und entscheiden, welche Ansicht gerendert werden soll.

Darüber hinaus verwenden wir das Konzept von Sessions und die Flask-Erweiterung "Flask-Login", um die Authentifizierung und Benutzerverwaltung zu implementieren. Dies ermöglicht es uns, Benutzer anzumelden und sicherzustellen, dass nur authentifizierte Benutzer auf bestimmte Seiten zugreifen können.