TH Brandenburg
Online Studiengang Medieninformatik
Fachbereich Informatik und Medien
Softwaretechnik
Prof. Dr-Ing. Martin Schafföner

Einsendeaufgabe 1: Requirements Engineering

Sommersemester 2021
Abgabetermin 27.04.2021

Mara Schulke
Matrikel-Nr. 20215853

Inhaltsverzeichnis

1 Aufgabenstellung

Folgende Aufgabenstellung wurde im Moodle-Kurs bekannt gegeben:

> Wählen Sie sich ein kleines Entwicklungsprojekt für das gesamte Studienmodul Softwaretechnik und verwenden Sie dieses Projekt für alle Lerneinheiten. Beispielsweise ein Tool zur Erfassung von Requirements oder ein anderes Tool, das Ihnen im Bereich der Softwaretechnik helfen kann, oder ein Tool, welches Sie schon immer mal bauen wollten. Im Verlauf des Moduls können Sie alle Themen wie beispielsweise Modellierung oder Testen anhand dieses Projekts praktisch erproben.
>
> Ermitteln und dokumentieren Sie die Requirements für ihr Projekt!
>
> - Formulieren Sie die Anforderungen als User Stories in der Sprache des Benutzers!
> - Formulieren Sie Abnahmeszenarien!
> - Reichern Sie weitere Informationen an, um die Qualitätskriterien für gute Anforderungen zu erfüllen!
>
> Stellen Sie weiterhin dar, wie, d.h. mit welchen Werkzeugen bzw. in welcher Struktur, Sie diese Requirements verwalten!
>
> Die Abgabe erfolgt in Form genau einer PDF-Datei. Falls sie Werkzeuge einsetzen, die keinen PDF-Export erlauben, fertigen Sie Screenshots an, die Sie mit der übrigen Beschreibung in eine PDF-Datei einfügen.
>
> Anmerkungen:
>
> - Der zeitliche Umfang dieser Einsendeaufgabe wird mit durchschnittlich 4 Stunden abgeschätzt
> - Eine Nachbearbeitung der abgegeben Lösung nach der Deadline ist nicht vorgesehen. Falls Sie Fragen zur Aufgabenstellung haben, fragen Sie diese also bitte im Vorfeld!

2 Projekt

Als Projekt für dieses Semester verwende ich eine Idee, die mir schon seit längerem im Kopf rumgeistert – einen kleinen, minimalistischen *Tiling Window Manager* für Linux bzw. X11, der deutlich flexibler und entwicklerfreundlicher ist als bestehende, vergleichbare Software.

2.1 Zusammenfassung der Domain

Für den Fall, dass Sie noch keinen Kontakt mit dieser Domain hatten, gehe ich im folgenden kurz darauf ein, welche Aufgaben ein *Tiling Window Manager* (kurz. *TWM* oder nur *WM*) typischerweise übernimmt. Die generelle Aufgabe eines WM's besteht hauptsächlich darin sich um die Kommunikation mit dem *Window-Server* (i.d.R. X11 oder Wayland auf Linux, Quartz Compositor auf MacOS) zu kümmern und die tatsächliche Anordnung der Fenster auf dem Bildschirm zu regeln (in Ebenen, Kacheln etc. – die Möglichkeiten sind nahe zu unbegrenzt). Typische *Window Manager* von z.B. MacOS können diverse, für uns als Endnutzer mittlerweile als üblich angesehene, Anordnungen wie Floating, Split-Screen und Fullscreen realisieren.

Ein *Tiling Window Manager* hat nun die Besonderheit, dass er anders als von Windows, MacOS oder diversen Linux Desktop-Umgebungen bekannt, die Aufteilung der Fenster automatisch

und bestmöglich regelt. Typischerweise hat ein *TWM* eine feste Konfiguration mit Layouts, in die er die Fenster einsortieren kann (z.B. "Master and Stack"). Somit muss sich der Anwender eines solchen *TWM's* (zumindest initial) nicht selber um die Anordnung seiner Fenster kümmern. Des Weiteren zählt die Möglichkeit, Keybindings zur Navigation oder Veränderung der Aufteilung zu definieren, zu den typischen Features eines *Tiling Window Managers*.

2.2 Projekt-Ziel

Das Problem von bestehenden *Window Managern* ist in der Regel deren Alter (bspw. ist das Projekt "Toms Window Manager" im Jahr 1987 entstanden). Aufgrund des Alters sind viele dieser *Window Manager* noch in C, C++ oder einer vergleichbaren Sprache verfasst worden und mit der Zeit immer weiter gewachsen. Dies macht es deutlich schwerer, als es sein müsste, grundlegende Änderungen, die nicht von den Entwicklern vorhergesehen waren, vorzunehmen.

Ziel ist es einen *minimalen Window Manager* zu entwickeln, der *durch eigenen Programm-Code konfiguriert* und erweitert werden kann. Dieses Konzept ist schon öfter Implementiert worden (als Inspiration für dieses Modell dienen die Projekte *XMonad*[1] und *DWM*[2]), allerdings sind diese Projekte leider meistens schlecht dokumentiert und haben i. d. R. unnötig komplexe *API's*. Zu den typischen Anwendern dieser *Window Manager* gehören (meiner subjektiven Erfahrung nach) Software Entwickler, System Administratoren und Power User. Diese drei Kategorien fasse ich im Folgenden zur Nutzerrolle "Konfigurierender" zusammen, da alle einen (für diesen Kontext) recht ähnlichen Kenntnisstand bzgl. der Domain haben.

3 Erfassung der Anforderungen

Zur Erfassung der Anforderungen verwende ich das Projektmanagement Feature von GitHub. Dieses hat zumindest im kleinen Rahmen diverse Vorteile gegenüber herkömmlicher Software für Agile Projekte (z.B. Jira von Atlassian). Unter anderem Folgende:

- Übersichtlicher durch kleineren Umfang und somit leichter zu Bedienen
- Direkte Integration mit dem Entwicklungsprozess (Issues können z.B. geschlossen werden sobald ein Feature gemerged wurde)
- Nutzer der Software können direkt Issues anlegen

Diese Entscheidung hat keine Endgültigkeit. Wenn es ein echtes Team mit einem klaren Vorgehensmodell gibt, können andere Tools besser geeignet sein. Außerdem sei erwähnt, dass Jira mit diversen Integrationen erweitert werden kann, um die oben beschriebene Funktionalität nachzurüsten. Zur geordneten Erfassung von Anforderungen, deren Priorisierung, Kategorisierung und Abnahme, reicht *GitHub Projects* allerdings vorerst vollkommen aus. Es gibt die Möglichkeit Sprint oder Kanban Boards anzulegen, diese zu automatisieren oder komplett eigene Prozesse zu integrieren.

[1] wurde 2007 erstmals veröffentlicht. Wurde in Haskell geschrieben und wird auch damit konfiguriert. Siehe `https://xmonad.org/` für weitere Informationen.

[2] ist ein äußerst kleiner (unter 2000 Zeilen Source-Code) Window Manager geschrieben in C. Er hat keine Konfigurationsdatei und wird durch Patchen des Codes konfiguriert. Siehe `https://dwm.suckless.org/` für weitere Informationen.

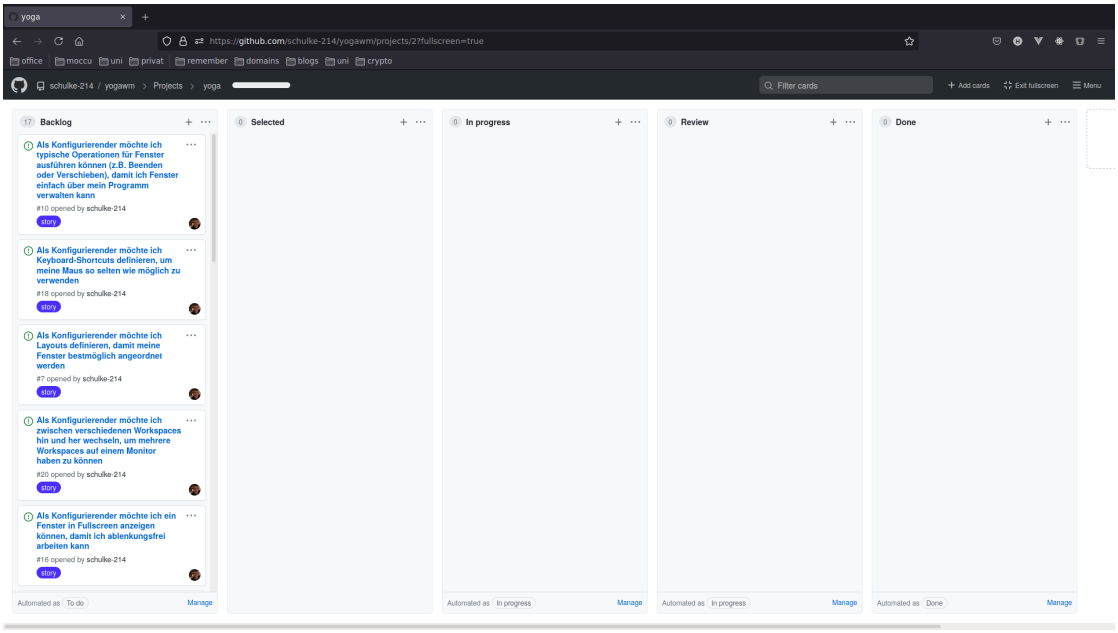

Abbildung 1: *GitHub Projects* Board

4 Anforderungen

Um die Lesbarkeit zu gewährleisten, habe ich hier eine Kopie des Backlogs aus GitHub Projects eingefügt. Einen Screenshot des Interfaces finden Sie weiter unten (Abbildung 2 auf Seite 8).

ID	Story
#10	Als Konfigurierender möchte ich typische Operationen für Fenster ausführen können (z.B. Beenden oder Verschieben), damit ich Fenster einfach über mein Programm verwalten kann.
#18	Als Konfigurierender möchte ich Keyboard-Shortcuts definieren, um meine Maus so selten wie möglich zu verwenden.
#7	Als Konfigurierender möchte ich Layouts definieren, damit meine Fenster bestmöglich angeordnet werden.
#20	Als Konfigurierender möchte ich zwischen verschiedenen Workspaces hin und her wechseln, um mehrere Workspaces auf einem Monitor haben zu können.
#16	Als Konfigurierender möchte ich ein Fenster in Fullscreen anzeigen können, damit ich ablenkungsfrei arbeiten kann.
#19	Als Konfigurierender möchte ich Fenster zwischen verschiedenen Workspaces verschieben, um die Kontrolle über die angezeigten Fenster je Workspace zu haben.
#17	Als Konfigurierender möchte ich Border für Fenster festlegen können, um zu sehen welches Fenster aktiv ist.
#9	Als Konfigurierender möchte ich eine Statusleiste einbinden können, um den Status, wie z.B. den momentanen Workspace, angezeigt zu kriegen.
#6	Als Konfigurierender möchte ich Regeln definieren, um Fenster einer Anwendung in einen Workspace einzusortieren.

#8	Als Konfigurierender möchte austauschbare Layouts, um eigene Layouts implementieren und verwenden zu können.
#12	Als Konfigurierender möchte ich eine gute Dokumentation der Schnittstellen-Funktionen, damit ich ohne weitreichendes Fachwissen eine Konfiguration erstellen kann.
#11	Als Konfigurierender möchte ich auf Metadaten von Fenstern zugreifen können, um in Abhängigkeit der Metadaten (z.B. Initiale Fenstergröße) Operationen auszuführen.
#4	Als Konfigurierender möchte ich eine Liste der angeschlossenen Bildschirme erhalten, um Regeln pro Bildschirm festzulegen.
#13	Als Konfigurierender möchte ich einfach Zugriff auf Community Erweiterungen haben, um so wenig wie möglich zu programmieren.
#5	Als Konfigurierender möchte ich beliebig viele Workspaces haben, um bestimmte Fenster in bestimmten Layouts anzuzeigen.
#15	Als Konfigurierender möchte ich einfach mit Maus-Events arbeiten, um die Maus als Alternative zur Tastatur verwenden.
#14	Als Konfigurierender möchte ich Hooks haben, damit ich die Funktionsweise grundlegend beeinflussen kann ohne die Schnittstelle zu Patchen.

Die Priorisierung der Anforderungen erfolgt relativ über die Reihenfolge im oben gezeigten Product-Backlog. Innerhalb von GitHub Projects können die einzelnen Stories noch über Flags kategorisiert werden (z.B. als Funktional oder als Epic)

4.1 Detaillierte Betrachtung einzelner Anforderungen

4.1.1 Details zur User Story #10

ID #10

Story Als Konfigurierender möchte ich typische Operationen für Fenster ausführen können (z.B. Beenden oder Verschieben), damit ich Fenster einfach über mein Programm verwalten kann.

Typ Funktional

Akzeptanzkriterien Szenario 1 - Fenster schließen

Gegeben einer gültigen Fenster-Referenz (z.B. durch ID)

Wenn der Nutzer dieses Fenster schließen will

Dann schließt sich dieses Fenster

Szenario 2 - Fenster verschieben

Gegeben einer gültigen Fenster-Referenz (z.B. durch ID)

Wenn der Nutzer dieses Fenster an eine Stelle (X, Y) bewegen will

Dann render dieses Fenster auf dem Bildschirm an die Koordinaten

Szenario 3 - Ungültiges Fenster

Gegeben einer ungültigen Fenster-Referenz (z.B. da das Programm abgestürzt ist)

Wenn der Nutzer versucht eine Operation (Beenden / Verschieben etc.) auszuführen

Dann gebe dem Nutzer eine Fehlermeldung zurück

4.1.2 Details zur User Story #20

ID #20

Story Als Konfigurierender möchte ich zwischen verschiedenen Workspaces hin und her wechseln, um mehrere Workspaces auf einem Monitor haben zu können.

Typ Funktional

Akzeptanzkriterien Szenario 1 - Workspace wechseln

Gegeben einer gültigen Workspace-ID

Wenn der Nutzer zu dem Workspace wechseln möchte

Dann verstecke alle Fenster im momentanen Workspace und zeige alle Fenster aus dem Workspace mit der gegebenen ID.

Szenario 2 - Zum letzten Workspace

Gegeben eines vorherigen Workspace-Wechsels

Wenn der Nutzer zum letzten Workspace möchte

Dann verstecke alle Fenster im momentanen Workspace und zeige alle Fenster aus dem vorherigen Workspace.

Szenario 3 - Ungültiger Workspace

Gegeben einer ungültigen Workspace-ID

Wenn der Nutzer zu dem Workspace wechseln möchte

Dann bleibe auf dem momentanen Workspace und gebe dem Nutzer eine Fehlermeldung zurück.

4.2 Weitere Merkmale von Anforderungen

Durch das Issue-Feature von GitHub werden viele Merkmale guter Anforderungen automatisch mit aufgenommen. Zum Beispiel werden in der Detail-Ansicht eines Issues (die als Story markiert werden kann) immer der Autor, die ID, das Erstellungsdatum, die Historie, ein Kommentarverlauf / bzw. eine Diskussion zu diesem Issue, der Status (Offen / Duplikat / Closed etc.) und ggf. die Person(en) die gerade an diesem Issue arbeitet/en festgehalten.

Außerdem kann man projektspezifische Templates für Issues anlegen, um z.B. Akzeptanzkriterien / Abnahmeszenarien immer in einem gewissen Umfang und einer ähnlichen Form zu erfassen.

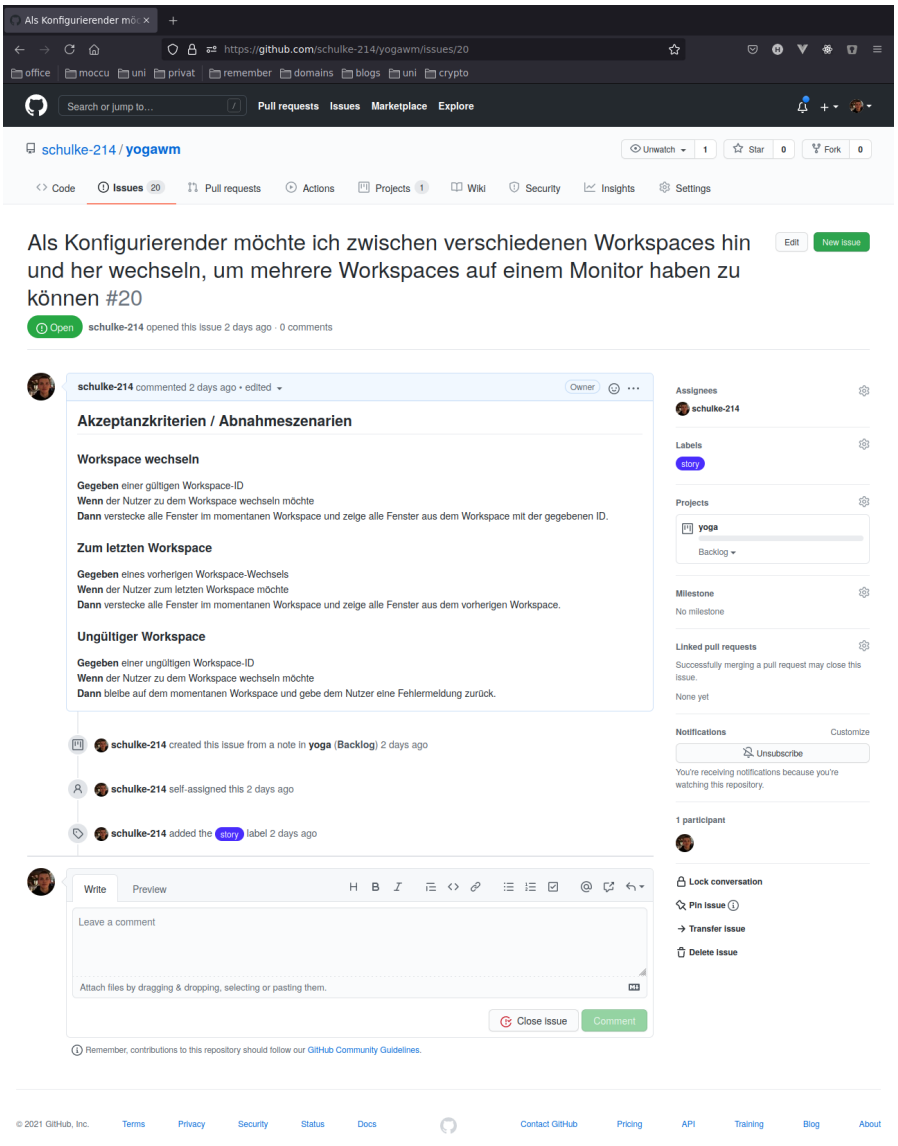

Abbildung 3: Detail-Ansicht einer Story

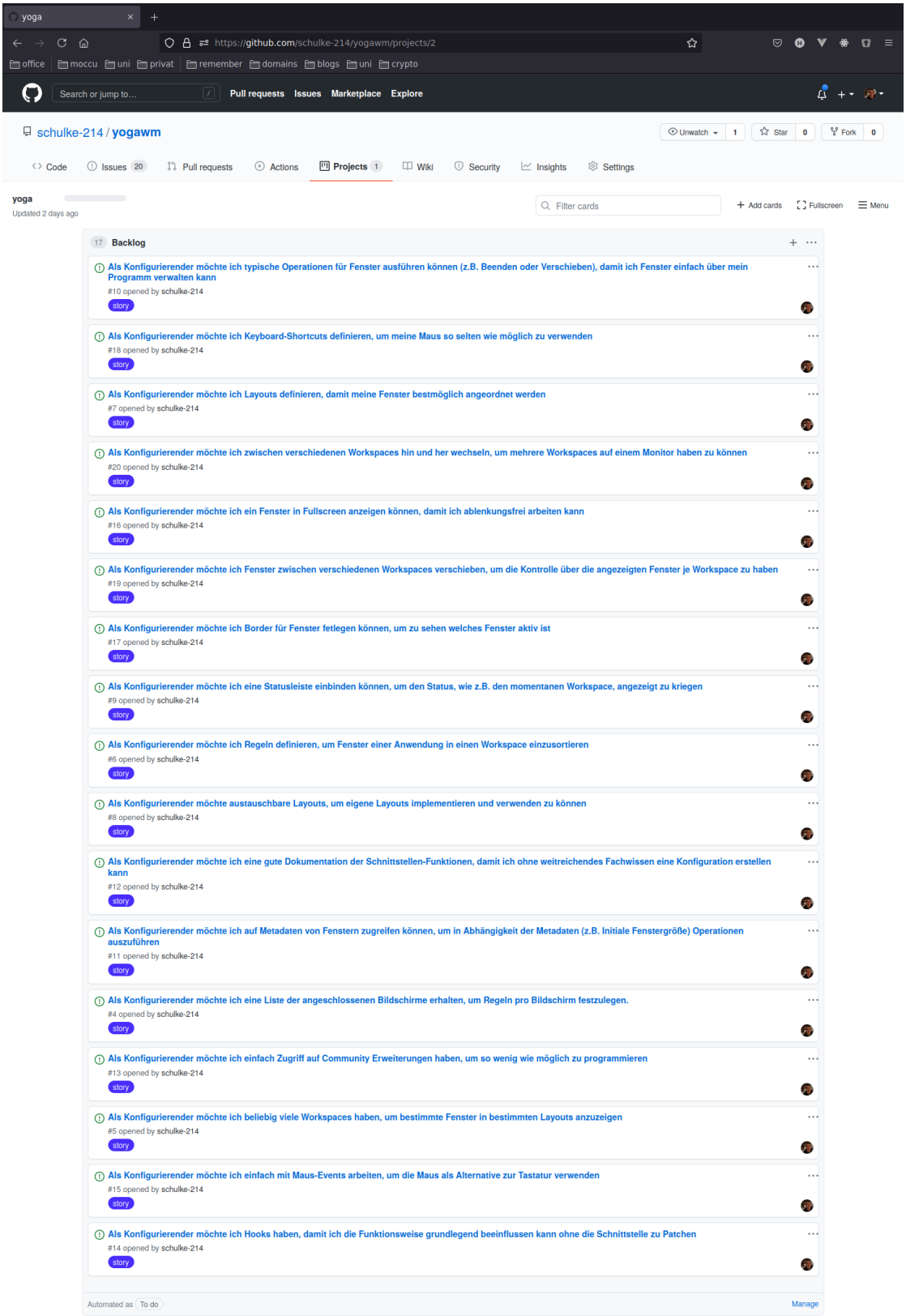

Abbildung 2: *Product Backlog*