Gitty

Anforderungsanalyse

|  |  |
| --- | --- |
| **Autor(en):** | Lian Studer |
| **Version:** | 1.0.0 |
| **Erstellt am:** | 3. Dezember 2020 |
| **Letzte Änderung:** | 3. Dezember 2020 |
| **Speicherort:** | Dokument2 |

Inhalt

[1 Revisionen 3](#_Toc531702604)

[2 Einleitung 4](#_Toc531702605)

[2.1 Systemidee 4](#_Toc531702606)

[2.2 Management Summary 4](#_Toc531702607)

[2.3 Projektteam 4](#_Toc531702608)

[2.4 Glossar 4](#_Toc531702609)

[3 Ausgangslage (IST), Problembereiche 5](#_Toc531702610)

[3.1 Beschreibung der Ausgangslage 5](#_Toc531702611)

[3.2 Problembereiche und Schwachstellen 5](#_Toc531702612)

[4 Ziele (SOLL) 6](#_Toc531702613)

[4.1 Beschreibung der Ziele 6](#_Toc531702614)

[4.2 Produktperspektive, Nutzen 6](#_Toc531702615)

[4.3 Zielkonflikte 6](#_Toc531702616)

[4.4 Abgrenzung 6](#_Toc531702617)

[5 Anforderungsanalyse 7](#_Toc531702618)

[5.1 Identifizierung der Akteure 7](#_Toc531702619)

[5.2 Anforderungskatalog 8](#_Toc531702620)

[5.2.1 F.REQ: Funktionale Anforderungen 8](#_Toc531702621)

[5.2.2 NF.REQ: Nichtfunktionale Anforderungen 9](#_Toc531702622)

[6 Systemablaufmodelle (Aktivitäten) 10](#_Toc531702623)

[6.1 Aktivität "DVD ausleihen" 10](#_Toc531702624)

[6.2 Aktivität «XY» 10](#_Toc531702625)

[7 Risiko-Analyse 11](#_Toc531702626)

[7.1 Risikokatalog 11](#_Toc531702627)

[8 Anhang 12](#_Toc531702628)

[8.1 Termine 12](#_Toc531702629)

[8.2 Referenzen 12](#_Toc531702630)

# Revisionen

|  |  |  |
| --- | --- | --- |
| **Datum** | **Name** | **Kommentar** |
| 03.12.2020 | Lian Studer | Dokument erstellt. |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

# Einleitung

## Systemidee

Die Idee hinter Gitty ist es, das klassische Git GUI mit dem CLI in Form eines Text User Interfaces (TUI) zu verbinden. Dadurch soll der Git Workflow für Leute, die gerne mit dem Git GUI arbeiten vereinfacht werden, da jeder Entwickler vermutlich immer irgendwo ein CMD / Terminal Fenster offen hat.

### Die wichtigsten Funktionen sind:

* Stage: Den Fortschritt eines Projekts stagen
* Commit: Gestagte Dateien commiten
* Push: Remote-Repositories hinzufügen und den Fortschritt pushen

## Management Summary

Die Version Control Software «Git», die mehrheitlich als Konsolenanwendung mit Befehlen funktioniert, soll mit Gitty in einer textbasierten Benutzeroberfläche benutzt werden können. Das heisst, dass die Software als graphische Oberfläche in der Konsole angezeigt wird.

## Projektteam

|  |  |  |
| --- | --- | --- |
| **Name** | **Funktion** | **Kontakt (Telefon, Mail...)** |
| Urs Nussbaumer | Projektleiter | 041 371 24 28 urs.nussbaumer@ict-bz.ch |
| Lian Studer | Entwickler | ln.studer@protonmail.ch |

## Glossar

Die im Projekt verwendeten Fachbegriffe und Abkürzungen in alphabetischer Reihenfolge.

|  |  |
| --- | --- |
| **Begriff** | **Beschreibung** |
| SRS | Software Requirement Specification (Anforderungsspezifikation nach IEEE) |
| SQL | Structured Query Language (Datenbankabfrage) |
| TUI | Text-basierte Benutzeroberfläche |
| GUI | Graphische Benutzeroberfläche |
| CLI | Befehlszeilen Programm |
| Git | Software zur Versionskontrolle |

# Ausgangslage (IST), Problembereiche

In diesem Kapitel wird die Ausgangslage beschrieben und es werden die Problembereiche identifiziert, die sich in der heutigen Situation zeigen.

## Beschreibung der Ausgangslage

Für die Git Software existieren unzählige GUIs zur erleichterten Bedienung der Version Control Software. Daneben gibt es das originale CLI von Linus Torvalds.

## Problembereiche und Schwachstellen

Der grosse Nachteil des Git-CLIs ist, dass die vielen Befehle für Beginner anfangs ein wenig zu kompliziert sein können. Ausserdem bevorzugen viele generell ein GUI. Jedoch muss man das GUI immer in einem separaten Fenster geöffnet haben, was auf Dauer mühsam ist.

# Ziele (SOLL)

In diesem Kapitel werden die übergeordneten Ziele beschrieben, die mit dem zu entwickelnden System erreicht werden sollen.

## Beschreibung der Ziele

Das Hauptziel der Lösung ist es, die Vor- und Nachteile der beiden Lösungen zu kombinieren und eine Art von GUI in der Konsole bereitzustellen. Man möchte mit der TUI Version von Git einen Kompromiss zwischen GUI und CLI bilden.

## Produktperspektive, Nutzen

Das Arbeiten mit einer graphischen Oberfläche macht Dinge oft einfacher und ein wenig übersichtlicher. Für Leute, die auf einem Linux basierten Betriebssystem arbeiten, ist das Terminal einer der wichtigsten Programme. Viele nutzen auch heute noch Texteditoren, die im Terminal laufen. Daher ist es für diese Leute einfacher Anwendungen direkt in der Befehlszeile zu verwenden.

# Anforderungsanalyse

Die Anforderungen an das zu entwickelnde System definieren alle zu erfüllenden Eigenschaften oder die zu erbringende Leistung, sowie allfällige technische Vorgaben und weitere Forderungen des Kunden im Zusammenhang mit dem (den) zu erstellenden Produkt(en).

## Anforderungskatalog

Der Anforderungskatalog ist eine priorisierte Liste, die alles enthält, was im zu entwickelnden Produkt enthalten sein soll. Es wird unterschieden zwischen funktionalen und nicht-funktionalen Anforderungen.

Jede Anforderung ist in Alltagssprache, in der Form einer "User-Story", formuliert, mit einer ID zur späteren Identifizierung versehen sowie mittels 3 verschiedener Kategorien priorisiert: 1 = hohe Prorität, 2 = mittlere Priorität, 3 = keine Priorität. Diese drei Prioritäten repräsentieren die Verpflichtungen "must", "should" und "nice to have".

Im Verlauf der Zeit können neue Anforderungen hinzukommen und/oder bestehende Anforderungen können wegfallen. Anforderungen, die wegfallen, sind im Dokument zu belassen und als ~~gestrichen~~ zu markieren.

Hinter einer Anforderung kann ihn eckigen Klammern in der Form [Z#] eine Zusatzinformation hinterlegt werden und mit [F#] bzw. [F#, F#, …] können eine oder mehrere offene Fragen referenziert werden.

### A: Funktionale Anforderungen

|  |  |  |
| --- | --- | --- |
| **ID** | **Anforderung** | **Prio** |
| A001 | Als Benutzer möchte ich einen File-Tree aller Dateien in der lokalen Repository sehen. | 1 |
| A002 | Als Benutzer möchte ich alle tracked Dateien und untracked Dateien visuell unterscheiden können. | 1 |
| A003 | Als Benutzer möchte ich eine zusätzliche dedizierte Git Konsole haben. | 1 |
| A004 | Als Benutzer möchte ich untracked Dateien per Keyboard-Action auswählen und tracken können. | 1 |
| A005 | Als Benutzer möchte ich alle getrackten Dateien commiten können. | 1 |
| A006 | Als Benutzer möchte ich Branches erstellen können. | 2 |
| A007 | Als Benutzer möchte ich zwischen Branches wechseln können. | 2 |
| A008 | Als Benutzer möchte ich Branches mergen können. | 2 |

### NF: Nichtfunktionale Anforderungen

|  |  |  |
| --- | --- | --- |
| **ID** | **Anforderung** | **Prio** |
| NF001 | Die Anwendung muss der Unix Philosophie folgen. | 1 |
| NF002 | Der Quellcode muss gut dokumentiert sein. | 1 |
| NF003 | Die Applikation muss mit «Make» installiert werden können. | 1 |
| NF004 | Die Git Konfiguration sollte in einer «.toml» oder «.yaml» Datei gespeichert werden, um sie manuell bearbeitbar zu machen. | 2 |

# Systemablaufmodelle (Aktivitäten)

Dieses Kapitel zeigt die wichtigsten und/oder komplexesten funktionalen Anforderungen in ihrem Ablauf. Dazu werden die einzelnen Aktivitätsschritte detailliert analysiert. Für die Darstellung der einzelnen Aktivitätsschritte werden Aktivitätsdiagramme nach UML verwendet.

Flussdiagramme / Aktivitätsdiagramme nötig für: TUI Konstruktion und Registration einer Benutzereingabe; Interaktion mit der Git API

# Anhang

## Termine

|  |  |  |
| --- | --- | --- |
| **Datum** | **Ziel, Beschreibung, Meilenstein** | **Verantwortlich** |
| 4.12.2020 | Sitzung mit Urs, Vorstellung des Projekts / Planung |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |