Belegarbeit GIT

Fabian Goerge
Sami Ede
Richard Peters
Jens Meise

28. November 2012

Inhaltsverzeichnis

FERTIGSTELLUNG BIS 09.12.!!!!!!!!!! und wehe nicht!!!!!!!!!!!!!
- PDF vollenden (gemeinsame Punkte bearbeiten)
- Präsentation fertigstellen (16.12. / 22.12.)

1 Einführung und Ziel der Arbeit

1.1 Einführung

...mit einem Merge kann man .....

Sami ab hier gehts los... Was ist Git? Git ist ein Versionsverwaltungssystem. Der Gedanke von Git ist das sich Daten im Laufe der Zeit ändern und Git bietet die Möglichkeit diese Änderungen zu Verwalten und festzuhalten. (Wer hat wann was und warum geändert?)

1.2 Ziel

Sami... Git modellieren, mit den Modellierungsarten auseinandersetzen

2 Textuelle Beschreibung

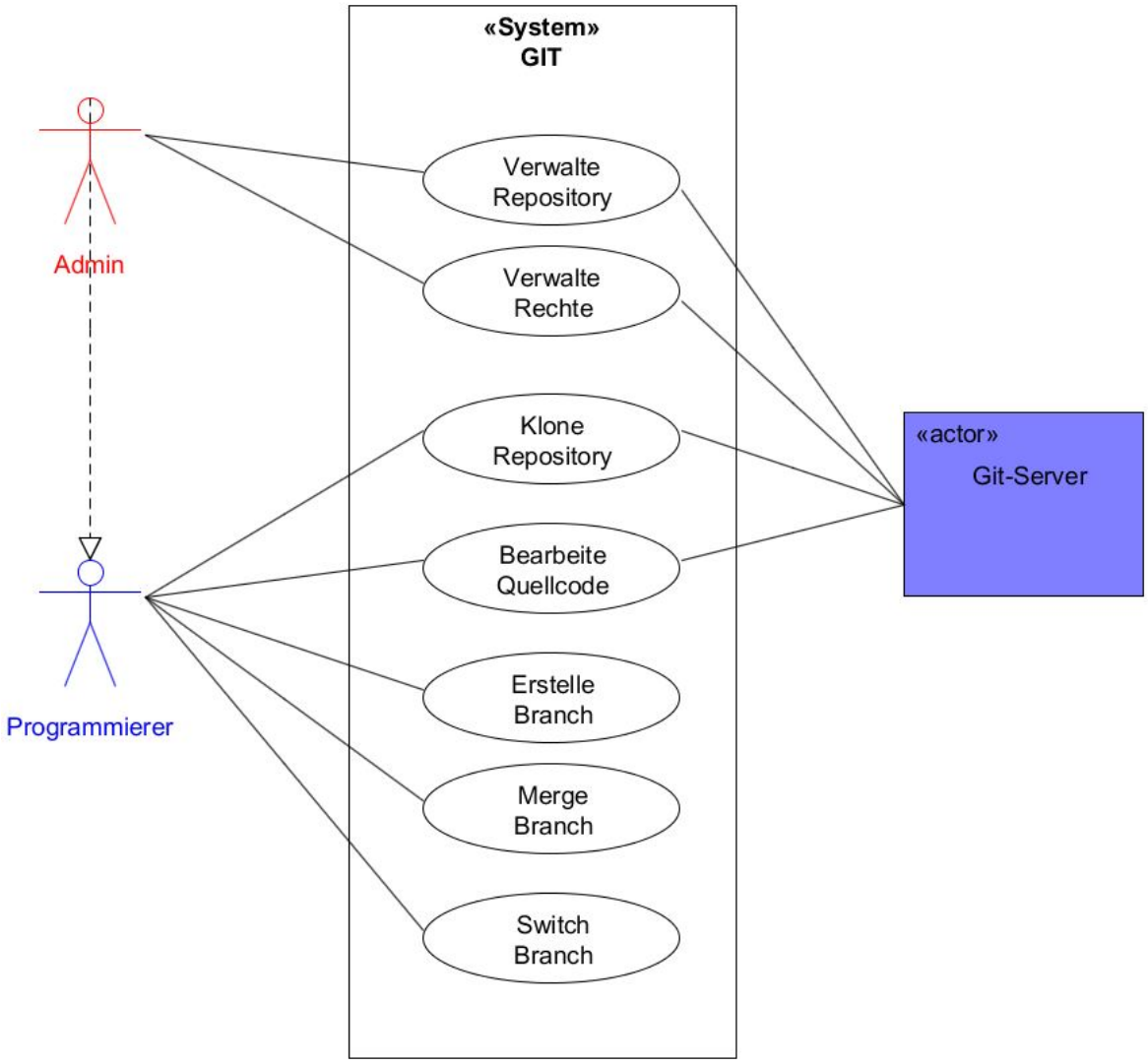
Git ist eine freie Software zur verteilten Versionsverwaltung von Dateien. Git hilft, wenn mehrere Programmierer an einem Projekt arbeiten, indem es verschiedene Features vereint:

- Git ermöglicht eine nicht-lineare Entwicklung; das heißt, es ist möglich, an verschiedenen „Positionen“ des Quelltextes zu arbeiten, und diese im Nachhinein automatisch zu vereinen („mergen“).
- des Weiteren bietet Git die Möglichkeit eines lokalen Arbeitens. Jeder User besitzt eine lokale Kopie des kompletten Repositories (inklusive der Versionsgeschichte), so dass auch ohne Anbindung in ein Netzwerk gearbeitet werden kann. Dies ermöglicht sehr schnelles Arbeiten! Dies bedeutet auch gleichzeitig Multi-Backups! Jeder User speichert die komplette Versionsgeschichte!
- Mehrere Möglichkeiten, Repositories abzugleichen; Git besitzt ein eigenes Protokoll, und unterstützt unteranderem http, https, ftp und rsync Übertragungen.
- Versionsgeschichte wird kryptografisch gesichert, so dass es nicht möglich ist, Änderungen an der Versionsgeschichte vorzunehmen, ohne dass sich der Name der Revision ändert.

Des Weiteren bietet Git ein Webinterface.

3 Modellierung

3.1 Use-Case-Modell



Dieses Use-Cases repräsentieren die Interaktion zwischen dem Programmierer (*Aktor*) und dem Git-System (*Subjekt*)

Der Programmierer benutzt das Git-System um ein bestimmtes Ziel zu erreichen. Der Git-Server (*Sekundäre Aktor*) wird von dem Git-System genutzt um eine Auswahl von Befehlen ausführen zu können.

Der Programmierer repräsentiert eine Personengruppe. An einem Projekt können beliebig viele Programmierer beteiligt seien. Der Admin erbt alle Fähigkeiten von dem Programmierer. Demnach kann der Admin auch alle anderen Interaktionen ausüben.

Im Use-Case sind die Möglichkeiten, Verwalte Repository, Verwalte Rechte, Klone Reopisitory, Bearbeite Quellecode, Erstelle Branch, Merge Branch und Switch Branch aufgeführt.

Eine detaillierte Beschreibung der Abläufe folgt in den Use-Case-Szenarien.

3.1.1 Use-Case-Szenarien (Storys)

- Happy day Szenarien und Optionale Verzweigungen (Abweichungen vom Happy Day) (Richard)

GIT Use – Case Szenarien

Use Case: Clone Repository
Kurzbeschreibung: Der Programmierer kopiert sämtliche Daten des Repositories auf seinen lokalen Rechner.
Vorbedingung: Ein Repository muss auf dem Server existieren
Primärer Aktor: Programmierer
Nachbedingung: Das Repository auf dem Rechner des Programmierers ist mit dem auf dem Server identisch
Erfolgsszenario:

Der Programmierer gibt das Kommando zum Klonen eines Repositories mit Serveradresse in die Shell ein. Das System verbindet sich mit dem Server. Das System überträgt das Repository vom Server auf den Rechner des Programmierers. Das System informiert den Programmierer über den Erfolg der Aktion. Das Use Case endet erfolgreich.

Use Case: Quellcode bearbeiten
Kurzbeschreibung: Der Programmierer möchte den Quellcode ändern. Dazu muss er das Repositorie auf den neuesten Stand bringen (‚pull'), den Quellcode bearbeiten, die Änderungen übernehmen (‚commit') und die Änderungen an den Server übertragen (‚Push')
Vorbedingung: Ein Repository muss sich auf dem Rechner und auf dem Server befinden.
Primärer Aktor: Programmierer
Nachbedingung: Die Änderungen des Programmierers werden registriert und in die History aufgenommen.
Erfolgsszenario:

Der Programmierer gibt das Kommando zum Holen von Änderungen in die Shell ein. Das System verbindet sich mit dem Server. Das System vergleicht den lokalen Zustand des Branches mit dem auf dem Server. Das System integriert (‚Merge') den Branch auf dem Server in den Branch auf dem Rechner. Das System informiert den Programmierer über den Erfolg der Aktion. Der Programmierer ändert den Quellcode Das System verfolgt die Änderungen. Der Programmierer gibt das Kommando zum Übernehmen (‚Commit') von Änderungen in die Shell ein mit einer Kurzbeschreibung seiner Änderungen. Das System übernimmt die Änderungen und trägt sie in die History des aktuellen Branches ein. Das System informiert den Programmierer über den Erfolg der Aktion. Der Programmierer gibt das Kommando zum Übertragen der Änderungen (‚Pull') auf den Server ein. Das System integriert den aktuellen Branch des Rechners in den entsprechenden Branch auf dem Server (‚Merge') Das System liefert dem Programmierer eine Erfolgsmeldung. Das Use Case endet erfolgreich.

Use Case: Branch erstellen
Kurzbeschreibung: Das System erstellt einen neuen Branch mit dem Zustand des Branches auf dem sich der Programmierer befindet.
Vorbedingung: Es muss ein Repository existieren.
Primärer Aktor: Programmierer
Nachbedingung: Es gibt einen neuen Branch.
Erfolgsszenario:

Der Programmierer gibt das Kommando zum Erstellen eines Branches in die Shell ein mit dem Namen des neuen Branches. Das System erzeugt einen neuen Branch. Das Use Case endet erfolgreich.

Use Case: Branch mergen
Kurzbeschreibung: Ein Branch wird in einen anderen Branch integriert
Vorbedingung: Es müssen 2 Branches existieren.
Primärer Aktor: Programmierer
Nachbedingung: Es gibt einen neuen Branch.

Erfolgsszenario:

Der Programmierer gibt das Kommando zum Mergen eines Branches in die Shell ein mit dem Namen des zu integrierenden Branches. Das System integriert den Branche in den aktiven Branche. Das Use Case endet erfolgreich.

Use Case: Branch switchen
Kurzbeschreibung: Der aktuelle Branch auf dem Rechner wird gewechselt (‚switch')
Vorbedingung: Es müssen mindestens 2 Branches existieren.
Primärer Aktor: Programmierer
Nachbedingung: Der aktuelle Branch wurde gewechselt.
Erfolgsszenario:

Der Programmierer gibt das Kommando zum Wechseln eines Branches (‚switch') in die Shell ein mit dem Namen des Zielbranches. Das System wechselt zum Zielbranch.Das Use Case endet erfolgreich.

Use Case: Repository verwalten
Kurzbeschreibung: Ein Repository wird verwaltet.
Vorbedingung: Keine – es kann auch erst bei der Verwaltung ein Repository angelegt werden.
Primärer Aktor: Admin
Nachbedingung: Repository verändert.
Erfolgsszenario:

Der Admin spezifiziert eine Verwaltungsoperation (löschen, erstellen umbenennen). Das System führt die entsprechende Aktion aus und informiert den Admin über den Erfolg der Aktion. Das Use Case endet erfolgreich.

Use Case: Rechte verwalten
Kurzbeschreibung: Der Admin kann Leserechte und ausführrecht für bestimmte Nutzergruppen verteilen.
Vorbedingung: Es muss mindestens eine Gruppe bestehen.
Primärer Aktor: Admin
Nachbedingung: Die Rechte wurden geändert.
Erfolgsszenario:

Der Admin gibt das Kommando zum Verwalten von Rechten in die Shell ein mit dem Namen der zu verwaltenden Gruppe. Das System listet die Rechte der ausgewählten Gruppe. Der Admin gibt entsprechende Kommandos zum Verwalten der Rechte ein. Das System informiert den Admin über eine erfolgreiche Ausführung der Aktion. Das Use Case endet erfolgreich.

Repository: Die Projektdaten und Informationen des Git-Systems (History, Branches, etc).

3.2 Domänenmodell

- Klassendiagramme, Objekte, Verknüpfungen, Abhängigkeiten (Jens, Richard)

3.3 Ausgewählte Zustandsdiagramme

- Verschiedene Prozesse beschreiben (Quelltext bearbeiten, Branches mergen) (Sami, Fabian)

3.4 Glossar

Branch engl. Zweig. Ein Entwicklungszweig in der Geschichte. 4

Klone engl. kopieren. 4

Merge engl. zusammenführen, Vereinigung von Branches. 2, 4, 5

Repository dt. Lager, Depot, Repositorium. enthält die Geschichte eines Projekts. Jeder Nutzer hat ein eigenes Repository mit der gesamten Geschichte. 4

Switch engl. wechseln. 4

- Fachwörter werden erklärt (Repository, Branch, bla...) (Gemeinsam im Laufe der Latexerstellung)

4 Erfahrung aus der Teamarbeit

gemeinsam gewählter Prozess... (Gemeinsam)

5 Schlussbetrachtung

Zusammenfassend kann man sagen das hier noch nicht so viel passiert ist ;) (Gemeinsam)