1 Seminar 1 - Werkzeuge der Softwareentwicklung

1.1 Aufgabe 1 - Werkzeugkategorien

- Texteditoren und integrierte Entwicklungsumgebungen (IDEs)
  - Zweck: Quellcode/Programmcode schreiben, entwickeln und debuggen
  - Beispiele: IntelliJ IDEA, Visual Studio 2022, Visual Studio Code, Atom
- Versionskontrollsysteme
  - Zweck: Verschiedene Versionen von Quellcode und Software verwalten, zusammenführen und dokumentieren
  - Beispiele: Git, Apache Subversion (SVN)
- Aufgabenmanagementsysteme
  - Zweck: Kapazitäten (zeitliche, finanzielle etc.) für einzelne Aufgaben der Softwareentwicklung managen
  - Beispiele: Trello
- Modellierungssysteme
  - Zweck: Erstellen von Modellen und Diagrammen, um den Entwurf von Software zu planen, die Software zu strukturieren und dokumentieren
  - Beispiele: Visual Paradigm, Umlet (beide arbeiten mit der Modellierungssprache UML)
- Programmiersprachen
  - Zweck: Überführung des Softwareentwurfs in von Computern ausführbare Befehle
  - Beispiele: Java, C/C++, Python, C#

1.2 Aufgabe 2 - GitHub

(a) Jedem GitHub-Code-Repository können über das Tool *GitHub Projects* beliebig viele *Projekte* hinzugefügt werden, welche jeweils einzeln geplant, durchgeführt und abgeschlossen werden können. Jedem Projekt werden Teammitglieder hinzugefügt und diesen Personen jeweils Rollen zugeordnet, bspw. *Owner*, *Member* oder *Billing Manager*.

Diese Projekte können weiter in einzelne Phasen unterteilt werden. Innerhalb dieser können sogenannte *Issues* und *Pull Requests* angeordnet und erledigt werden.

Issues sind ein Mittel, welches GitHub bereitstellt, um Ideen, Feedback, Aufgaben oder Bugs in vorhandenem Code zu dokumentieren und zu verwalten. Sie werden projektspezifisch angelegt und können durch entsprechende Änderungen am Projekt, welche z.B. die Idee umsetzen oder den Bug beheben, abgeschlossen werden.

Pull Requests sind ein von GitHub bereitgestellter Weg, um Änderungen an einem Projekt zu dokumentieren und zu verwalten. Wenn ein Teammitglied eine Version des Projektes (einen *branch*) verändert, so wird dieser in der Regel nicht direkt mit der aktuellen Hauptversion (dem *base branch*) verschmolzen. Zunächst wird ein Pull Request geöffnet, in welchem die vorgenommenen Änderungen beschrieben sind, mit anderen Beteiligten diskutiert werden und weiter angepasst werden können.

Im Verlaufe des Projekts wird dessen Fortschritt direkt auf GitHub angezeigt, wodurch es beispielsweise vereinfacht wird, den aktuellen Stand eines Projektes an Vorgesetzte zu kommunizieren. Außerdem können jedem Projekt Meilensteine hinzugefügt werden.

Weiterhin ist es möglich, einzelne Aufgaben (*tasks*) anzulegen und sie verschiedenen Teammitgliedern zuzuweisen. Dadurch wird es möglich, große Aufgaben aufzuteilen und den Fortschritt einfacher zu kontrollieren. Bei jeder Änderung eines Tasks werden dessen Historie und Entwicklung automatisch aktualisiert. So können fehlerhafte Änderungen auch leicht rückgängig gemacht werden.

Außerdem können an jedem Projekt von jedem Teammitglied Notizen angebracht werden. Diese können z.B. Ideen bezüglich der Entwicklung des Projektes enthalten oder andere Teammitglieder markieren. Darüber hinaus ist es möglich, mittels sogenannter *Reviews* Dateien untereinander zu teilen.

Schließlich ist es möglich, jedem Projekt einige Verhaltensregeln (*Code of Conduct*) hinzuzufügen, an die sich jedes Teammitglied halten soll. Auf diese Weise wird die Arbeit an dem Projekt strukturiert und jedes Mitglied des Teams erhält so eine Richtlinie, an die sich gehalten werden kann.

(b) Welche Personengruppe können am Prozess beteiligt sein?
Es sind Softwarearchitekten, Ressourcenmanager (Finanzmanager), Projektplaner, Softwareentwickler, Kontrollinstanzen (Tester), Sicherheitsbeauftragter und Analytiker an dem Prozess der Softwareentwicklung mit GitHub beteiligt sein.

1.3 Aufgabe 3 - Versionsverwaltung

(a) Durch eine Versionsverwaltung ist es möglich, vergangene Versionen eines Projektes wiederherzustellen, falls die aktuelle Version fehlerbehaftet ist. Weiterhin ermöglicht es in großen Projekten, dass einzelne Teammitglieder separat an einem Projekt arbeiten können (in *branches*), ohne sich gegenseitig zu behindern. Anschließend ist es möglich, über die Versionsverwaltung alle gemachten Änderungen zusammenzuführen bzw. zu verschmelzen (*merge*). Weiterhin kann eingesehen werden, welche Person wann welche Datei(en) geändert hat und aus welchen Gründen diese Änderung vorgenommen wurde. Außerdem können darüber Anmerkungen gemacht werden, bspw. von für das Team verantwortliche Personen, um dem Programmierer Feedback zu geben.

(b)
- *Branch*: Ein Nebenstrang des Programms an dem ein Entwickler getrennt von anderen Entwicklern arbeiten kann. Dieser Strang kann später zum Main- bzw. Base-Branch hinzugefügt werden, welcher die Hauptversion des Quelltextes des Programms darstellt.
- *Checkout*: Dieser Git-Befehl ermöglich es, zwischen verschiedenen Branches zu wechseln bzw. neue zu erstellen.
- *Pull-Request & Merge*: Ein *Pull-Request* ist ein Mittel, um die Zusammenführung eines Branches mit dem Base-Branch anzufragen. Die tatsächliche Zusammenführung der Branches wird *Merge* genannt.
- *Tags*: Ein Verweis auf einen festgelegten Punkt in der Historie eines Git-Projektes. Üblicherweise werden sie genutzt, um Release-Versionen von Software zu markieren. Ein *Tag* kann als ein unveränderlicher *Branch* betrachtet werden.

(c) Zunächst muss ein Main-Branch angelegt werden. Für jede geplante Änderung bzw. Erweiterung des vorhandenen Quelltextes sollte dann ein weitere Branch erstellt werden, in

dem die Änderungen sicher vorgenommen werden können, ohne die funktionierende Version des Quelltextes im Main-Branch zu verändern. Anschließend kann ein Pull-Request eröffnet werden, wenn alle gewünschten Änderungen bzw. Erweiterungen innerhalb eines Branches vorgenommen wurden. Diese Pull-Requests werden dann von anderen Entwicklern begutachtet, ggf. noch nachgebessert und schließlich durch eienn Merge mit dem Main-Branch zusammengeführt.

1.4 Aufgabe 4 - Entwicklungsumgebungen

- *Unterstützung von Versionskontrolle*: Viele IDEs unterstützen die automatisierte Ausführung von Git-Kommandos für das bearbeitete Repository, u.a.:
  - *commit*: Veränderungen am Quelltext können erst lokal und dann direkt auf dem Versionierungsserver vorgenommen werden.
  - *update*: Es ist möglich, alle Meta-Informationen des Projektes (aktuelle Branches, commits etc.) vom Versionierungsserver zu kopieren.
  - Ausführung von *Pull-Requests*
- automatisches Hinzufügen/Entfernen von Dateien
- meistens: eine Einfärbung des Quelltextes zur besseren Lesbarkeit (*Syntax Highlighting*)
- *Code-Vervollständigung*, um Tipparbeit zu verringern; damit inbegriffen: Vorschlagen vergebener Namen (also von bereits deklarierten Variablen, Funktionen etc.)
- einfache automatische *Code-Generierung* (z.B. for-Schleifen)
- Projekte erstellen, kompilieren und ausführen
- *Debugging*, z.B. durch das Hinzufügen von Haltepunkten (*breakpoints*) im Quellcode, um die Fehlerfindung zu erleichtern

1.5 Aufgabe 5 - Dokumentation

(a) Nachfolgend sind verschiedene Arten von Dokumentationen und dazu passende Programme aufgelistet.
- Anforderungsdokumentation
  - Modern Requirements
  - Jama Software
  - Visur
- Architekturdokumentation
  - UML (z.B. mittels des Tools Visual Paradigm)
  - Microsoft Visio
  - Omnigraffle
- Entwicklerdokumentation
  - GitHub
  - Apiary
  - Read the Docs

- Endnutzer-Dokumentation
  - Whatfix
  - Bit.ai
  - ProProfs
  - Dropbox Paper

(b) Modellierung und UML helfen dabei, die Dokumentation zu strukturieren und in ihr keine Komponente der Software unerwähnt zu lassen, da sie einen Überblick darüber geben, welche Komponenten die Software ausmachen, in ihr enthalten sind und wie diese miteinander wechselwirken. Insbesondere Letzteres ist ein zentraler Bestandteil des Entwurfs der Softwarearchitektur bzw. -struktur, welche für alle beteiligten Personen wichtig ist, um die Software sauber entwickeln zu können.