ICT Berufsbildung Zentralschweiz

Modul 318

Analysieren und objektbasiert programmieren mit Komponenten

Autoren: Hanspeter Stalder / Stefan Weber / Urs Nussbaumer

Version: 1.1

Datum: 12.04.2017

ICT Berufsbildung Zentralschweiz

1 Inhalt

1 Einführung

> *Ich habe mir immer gewünscht, dass mein Computer so leicht zu bedienen ist wie mein Telefon; mein Wunsch ging in Erfüllung: mein Telefon kann ich jetzt auch nicht mehr bedienen.*
>
> Bjarne Stroustrup

Als Programmierer müssen wir immer mehrere Ziele verfolgen. Unsere Programme sollten benutzerfreundlich sein, sie müssen die Kundenanforderungen erfüllen und sie sollten einfach zu warten und zu erweitern sein. Wir sollten sie kostengünstig erstellen können und nicht zuletzt wollen wir Spass dabeihaben.

Kann Euch dieser ÜK dabei helfen ein besserer, benutzer-naher und effizienter(er) Entwickler zu werden? Einer, der bei der Arbeit auch noch Spass hat?

Wir haben versucht den Kurs auf diese Ziele auszurichten - im Folgenden werdet Ihr sehen was es mit diesem Kurs auf sich hat.

Wenn man die Modul Identifikation und die handlungsnotwendigen Kenntnisse des Modules anschaut, sieht er auf den ersten Blick wie jeder andere ÜK aus. Schaut man aber genauer hin, dann wird klar, dass dieser Kurs extrem viel Potential hat.

Es geht um die Analyse einer Aufgabenstellung, es geht um die Gestaltung einer Benutzeroberfläche, also um GUI-Ergonomie. Und es geht um Programmierung, um Eventsysteme und sogar das Konzept der Objektorientierung wird gestreift. Auch die Qualität und das Testen werden thematisiert und wir werden Komponenten-Bibliotheken benutzen, die uns von den modernen Frameworks und Entwicklungs-umgebungen zur Verfügung gestellt werden.

Last but not least werden wir auch noch lernen, wie man Source Code mit GIT verwalten kann.

Wir hoffen, dass Euch dieser ÜK schmecken wird.

2 Überblick

Folgende Themen werden wir anschauen und in der Praxis üben.

2.1 Analyse und Design

Wie werden lernen wie man Anforderungen analysiert und diese in klare Entwicklungsaufgaben (Use Cases, User Stories) unterteilt. Wir werden Methoden kennenlernen zur Modellierung und Visualisierung von Programmabläufen.

Gutes Testing ist wichtig; es hilft eine qualitativ gute Software zu erhalten. Dennoch müssen wir verstehen, dass Qualität nicht in die Software “hinein getestet” werden kann, sondern Qualität entsteht vor und während der Entwicklung der Software, das heisst, täglich und in jedem Moment unserer Arbeit.

Es gibt viele Hilfsmittel und Möglichkeiten, wenn es darum geht, Software zu designen. Grundlegende Dinge dabei sind:

- Welche Anwendungsfälle (Funktionen) soll meine Software abdecken? → Use Case Diagramm
- Wie sieht der Kontext der Software aus? → System Context Diagram
- Wie ist der Ablauf einer Funktion? Wie sehen Kontroll- und Datenflüsse aus? → Aktivitätsdiagramm
- Welche Klassen brauche ich für die Lösung meines Problems? → Klassendiagramm
- Wie interagieren meine Klassen untereinander? → Sequenzdiagramm

Die Architektur und das Design sind auch Aufgabe des ÜK318.

2.2 Ergonomie und GUI Gestaltung

Wir werden sehen, auf was man bei der Gestaltung von Benutzeroberflächen achten muss. Und wir werden ein GUI für die Projektaufgabe entwerfen.

2.3 Testing

Das Testing ist integraler Teil der Softwareentwicklung. Dabei sollte man nicht erst am Schluss über das Testing nachdenken, sondern ganz im Gegenteil schon ganz früh in einem Projekt. Wenn man sich schon früh Gedanken über das Testing macht führt dies zu besserer Software, weil man als Tester die richtigen Fragen stellt und so zu einer genaueren Anforderungsanalyse und schliesslich zu einer besseren Lösung kommt.

3 Die Projektarbeit

In diesem ÜK geht es darum, eine Applikation zu programmieren, welche die Fahrplandaten des Schweizerischen öffentlichen Verkehrs benutzt. Mit der Applikation soll es möglich sein, Verkehrsverbindungen zwischen zwei Stationen zu suchen. Aus der Bewertung dieser Projektarbeit ergibt sich die ÜK-Note.

3.1 Anforderungen

Im Folgenden hat der Kunde die Anforderungen in der Reihenfolge ihrer Priorität niedergeschrieben.

Prioritäten: 1 = must / 2 = should / 3 = nice to have

ID	Beschreibung	Priorität
A001	Als ÖV-Benutzer möchte ich Start- und Endstation mittels Textsuche suchen können, damit ich nicht alle Stationsnamen auswendig lernen muss.	1
A002	Als ÖV-Benutzer möchte ich die aktuellen, d.h. mindestens die nächsten vier bis fünf Verbindungen zwischen den beiden gefundenen und ausgewählten Stationen sehen, damit ich weiss wann ich zur Station muss, um den für mich idealen Anschluss zu erwischen.	1
A003	Als ÖV-Benutzer möchte ich sehen, welche Verbindungen ab einer bestimmten Station vorhanden sind, damit ich bei mir zuhause eine Art Abfahrtstafel haben kann.	1
A004	Als ÖV-Benutzer möchte ich, dass schon während meiner Eingabe erste Such-Resultate erscheinen, damit ich effizienter nach Stationen suchen kann.	2
A005	ALs ÖV-Benutzer möchte ich nicht nur aktuelle Verbindungen suchen können, sondern auch solche zu einem beliebigen anderen Zeitpunkt, damit ich zukünftige Reisen planen kann.	2
A006	Als ÖV-Benutzer möchte ich sehen, wo sich eine Station befindet, damit ich mir besser vorstellen kann, wie die Situation vor Ort aussieht.	3
A007	Als ÖV-Benutzer möchte Stationen finden, die sich ganz in der Nähe meiner aktuellen Position befinden, damit ich schnell einen Anschluss erreichen kann.	3
A008	Ich möchte meine gefundenen Resultate via Mail weiterleiten können, damit auch andere von meinen Recherchen profitieren können.	3

3.2 Qualität

Software hat spezifische Qualitätsmerkmale.

1. **Funktionalität**
   Hier wird geprüft, ob die Software richtig funktioniert.
2. **Zuverlässigkeit**
   Hier schaut man ob die Software zuverlässig funktioniert, was in Fehler-Situationen passiert, ob verständliche Fehlermeldungen angezeigt werden, ob sich Daten wiederherstellen lassen usw.
3. **Benutzbarkeit**
   Kann man die Software einfach bedienen? Ist die Benutzeroberfläche übersichtlich und selbsterklärend gestaltet? Versteht man die Funktionen?
4. **Effizienz**
   Definiert die Performance (das Zeitverhalten) der Applikation, den Speicherverbrauch usw.
5. **Wartbarkeit**
   Kann man die Software einfach/problemlos erweitern? Wie einfach ist es, Fehler einzugrenzen und zu lokalisieren? Wie stabil verhält sich die Software bei Änderungen (Stichwort: Side Effects)? Ist der Source Code gut und verständlich kommentiert? Wurden die Programmierrichtlinien durchgängig eingehalten?
6. **Portierbarkeit**
   Kann man die Software einfach installieren? Kann sie mit anderen Programmen koexistieren? Welche Auswirkungen hat die Installation der Software auf andere Programme? usw.

Die Projektarbeit wird unter anderem anhand dieser Qualitätsmerkmale bewertet. Um klarer zu sehen was hier relevant ist, werden nachfolgend die Qualitätsziele definiert.

3.3 Qualitätsziele

Funktionalität:

- Die Funktionalität der Software soll den Anforderungen entsprechen, welche laut Dokumentation umgesetzt wurden.
- Die Funktionen sollen korrekt umgesetzt sein.
- Die richtige Funktionsweise soll durch Testfälle sichergestellt sein.

Zuverlässigkeit:

- Die Software soll zuverlässig funktionieren und tun, was man von ihr erwartet.
- Die Software soll in Fehlerfällen den Benutzer korrekt (mit verständlichen Fehlermeldungen) informieren ohne abzustürzen.
- Nach Auftreten eines Fehlers sollte die Software, wenn möglich, normal weiterlaufen.

Benutzbarkeit:

- Die Bedienung der Software soll selbsterklärend sein.
- Benutzereingaben sollen validiert werden.
- Die grafische Benutzeroberfläche (GUI) soll übersichtlich gestaltet sein.
- Das GUI soll sich an UI Standards halten.

Wartbarkeit:

- Der Source Code soll sich an die Programmierrichtlinien (Coding Guidelines) halten.
- Variablen, Klassen, Methoden und andere Elemente sollen sprechende Namen haben.
- Der Source Code soll verständlich kommentiert sein, insbesondere sollen alle öffentlichen Methoden inklusive ihrer Parameter kommentiert sein.
- Es soll einfach und problemlos möglich sein, Änderungen an der Software zu machen. Dazu gehört: kein repetitiver Code (copy paste), kein Spaghetticode, Beachtung des Kohäsionszprinzips und eine möglichst lose Kopplung der einzelnen Module und Klassen.

Portierbarkeit:

- Die Software soll einfach zu installieren sein.
- Die Software soll auch einfach wieder entfernt werden können.
- Die Software soll nach der Installation auch auf einem Rechner ohne Entwicklungsumgebung lauffähig sein.
- Zusammen mit der Software soll eine Installationsanleitung ausgeliefert werden.

3.4 Dokumentation

Es wird keine Dokumentation im eigentlichen Sinne verlangt. Es braucht jedoch im Minimum ein Dokument, das die folgenden Informationen enthält.

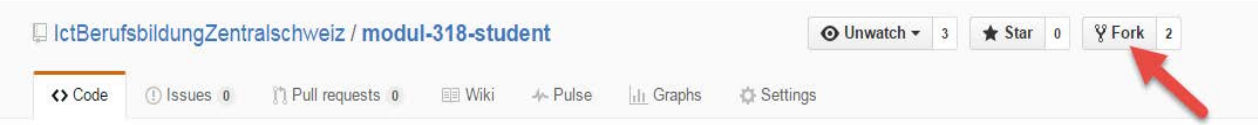
- Autor, Datum.
- Eine Einleitung (Management Summary).
- Zweck des Dokuments.
- Was (d.h. welche Funktionen) wurde umgesetzt?
- Falls bestimmte Funktionen nur teilweise umgesetzt wurden: Welche? Welcher Teil der Funktionalität fehlt noch? Bekannte Fehler/Bugs?
- Use Cases und Aktivitätendiagramme für alle umgesetzten Anforderungen aber mindestens für die mit Priorität 1 klassifizierten Anforderungen.
- Testfälle (Systemtests), verständlich und **eindeutig nachvollziehbar** geschrieben, so dass ein Tester diese ohne weiteren Erklärungen durchführen kann.
- Installationsanleitung: Wie wird die Software installiert? Wie wird die Software deinstalliert?
- Andere spannende Informationen für die Bewertung.

Die Dokumentation soll sauber gestaltet, inhaltlich korrekt und übersichtlich strukturiert sein. Ein Inhaltsverzeichnis soll vorhanden sein. Achte auf Typographie und Rechtschreibung!

4 Erste Schritte

Wie komme ich zu meiner Visual Studio Solution?

1. Falls Du noch keinen Account bei github habt: erstelle einen.
2. Es gibt ein Repository auf github:
   https://github.com/IctBerufsbildungZentralschweiz/modul-318-student
3. Forke dieses Repository



4. Bereite ein Directory auf deinem PC vor, wo du entwickeln willst.
5. Klone **dein** Repository an diesen Platz:

```
git clone https://github.com/YOUR-USERNAME/YOUR-REPOSITORY
```

Alles weitere siehst du im folgenden Kapitel «GIT» oder auf den Hilfeseiten von Github.

5 Versionskontrolle mit Git

Den Source Code deiner Software wirst du mit Git verwalten. Hier findest du eine kurze Anleitung wie das grundsätzlich funktioniert. Siehe auch: https://rogerdudler.github.io/git-guide/index.de.html.

5.1 Ein Repository auschecken

Erstelle eine Arbeitskopie, indem du folgenden Befehl ausführst:

```
git clone /pfad/zum/repository
```

Falls du ein entferntes Repository verwendest, benutze:

```
git clone benutzername@host:/pfad/zum/repository
```

5.2 Workflow

Dein lokales Repository besteht aus drei "Instanzen", die von Git verwaltet werden. Die erste Instanz ist deine Arbeitskopie, welche die «echten» Dateien enthält. Die zweite ist der Index (Stage), welcher als Zwischenstufe agiert und zu guter Letzt ist da noch der HEAD, der immer auf deinen letzten Commit zeigt.

Du kannst Änderungen vorschlagen (zum Index hinzufügen) mit

```
git add <dateiname>
git add *
```

Der nächste Schritt im Git-Workflow ist es, deine Änderungen zu bestätigen mit:

```
git commit -m "Commit-Nachricht"
```

Jetzt befindet sich die Änderung im HEAD, jedoch noch nicht im entfernten Repository.

5.3 Änderungen hochladen

Die Änderungen sind jetzt im HEAD deines lokalen Repositories. Um die Änderungen an dein entferntes Repository zu senden, führe folgende Anweisung aus:

```
git push origin master
```

Du kannst *master* auch mit einem beliebigen anderen Branch ersetzen (mehr über Branches erfährst du später).

5.4 fetch & merge

Um dein lokales Repository mit den neuesten Änderungen aus dem entfernten Repo zu aktualisieren, verwende:

```
git pull
```

Mache dies in deiner Arbeitskopie, um die Änderungen erst herunterzuladen (fetch) und dann mit deinem Stand zusammenzuführen (merge).

Wenn du einen anderen Branch mit deinem aktuellen (z.B. master) zusammenführen willst, benutze:

```
git merge <branch>
```

In beiden Fällen versucht Git die Änderungen automatisch zusammenzuführen. Unglücklicherweise ist dies nicht immer möglich und kann in Konflikten enden (merge conflict). Du bist verantwortlich, diese Konflikte durch manuelles Editieren der betroffenen Dateien zu lösen. Bist du damit fertig, musst du das GIT mit folgendem Befehl mitteilen:

```
git add <dateiname>
```

Bevor du Änderungen zusammenführst, kannst du dir die Differenzen auch anschauen:

```
git diff <quell_branch> <ziel_branch>
```

5.5 Tagging

Es wird empfohlen, für Software Release-Tags zu verwenden. Dies ist ein bekanntes Konzept, das es auch schon mit SVN gab. Du kannst einen neuen Tag namens "1.0.0" mit folgendem Befehl erstellen:

```
git tag 1.0.0 1b2e1d63ff
```

1b2e1d63ff steht für die ersten 10 Zeichen der Commit-Id, die du mit deinem Tag referenzieren möchtest. Du erhältst die Liste der Commit-IDs mit:

```
git log
```

Du kannst auch weniger Zeichen verwenden, es muss einfach eindeutig sein.

5.6 Änderungen rückgängig machen

Falls du mal etwas falsch machst - was natürlich nie passiert ;) - kannst du die lokalen Dateien mit:

```
git checkout -- <filename>
```

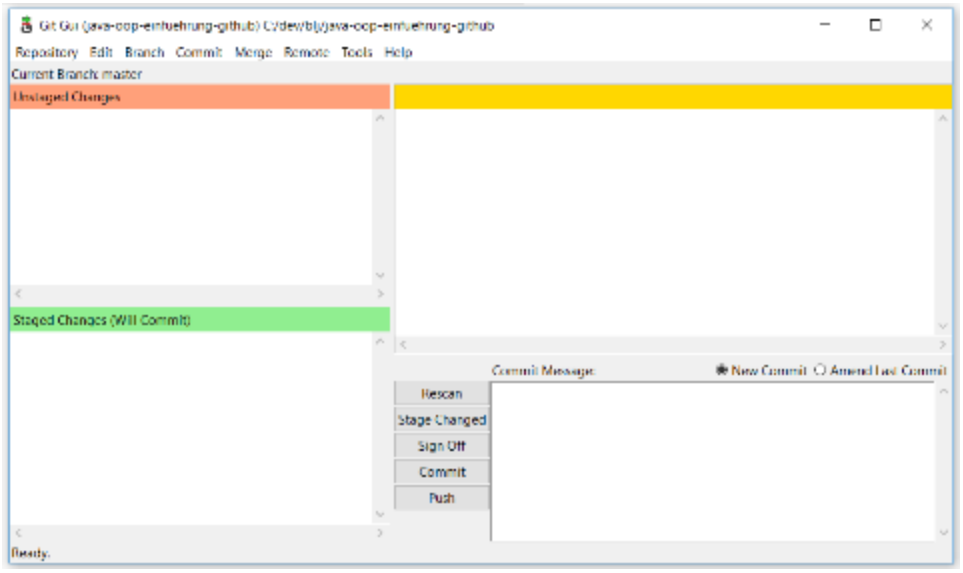
auf den letzten Stand im HEAD zurücksetzen. Änderungen, die du bereits zum Index hinzugefügt hast, bleiben bestehen.

Wenn du aber deine lokalen Änderungen komplett entfernen möchtest, holst du dir den letzten Stand vom entfernten Repository mit folgenden Befehlen:

```
git fetch origin
git reset --hard origin/master
```

5.7 Git GUI

Falls du lieber mit einer grafischen Benutzeroberfläche arbeitest, als mit der Kommandozeile, kannst du die wichtigsten Arbeitsschritte des Git-Workflows auch mit Git GUI erledigen. Git GUI wird standardmässig zusammen mit Git auf deinem Rechner installiert.



6 Wie wird bewertet?

Folgende Bereiche werden bewertet:

1. Funktionalität (mindestens die Anforderungen A001 - A003)
    a. Anforderungen erfüllt?
2. Quellcode
    a. Kann er von GitHub geholt werden?
    b. Kann er kompiliert werden?
    c. Clean Code und Co.
    d. Codequalität.
3. Benutzerfreundlichkeit
    a. GUI (Entwurf, Implementation, Übersichtlichkeit, Funktionalität).
    b. Werden Benutzereingaben validiert?
    c. Wurde Fehlerbewusst programmiert? (keine Abstürze, verständliche Fehlermeldungen)
    d. Installation und Deinstallation.
4. Dokumentation
    a. Testing (Testfälle).
    b. Analyse, Architektur, Design (Plan und Umsetzung).
    c. Struktur, Gesamteindruck

Diese vier Bereiche werden gleichwertig zu je 25% gewichtet. Detailliertere Anhaltspunkte sind im Kapitel Qualitätsziele erwähnt sind.

7 Systemvoraussetzungen

Entwicklungsumgebung

- Windows 64-Bit Betriebssystem
- Visual Studio Community 2017: https://www.visualstudio.com/de/downloads/
- Git : https://git-scm.com/downloads

Optional

- Style Cop Plugin für Visual Studio
- Git Plugin für Visual Studio

Nützliche Tools

- Visual Paradigm Community Edition: https://www.visual-paradigm.com/download/community.jsp
- UML Tool (Klassendiagramme): http://www.nomnoml.com/
- UML Diagrams: http://plantuml.com/
- Sequence Diagram: https://www.websequencediagrams.com/
- Creatly (Demo-Modus): https://creately.com/diagram-products#online