Übungen zur Vorlesung

Softwaretechnologie

-Wintersemester 2013/2014-

Dr. Günter Kniesel

Übungsblatt 1 - Lösungshilfe

.

Aufgabe 1. *Erste Schritte mit Subversion* (5 Punkte)

a) Im Repository Ihrer Gruppe befindet sich ein Projekt namens *HelloTutor*. Checken Sie das Verzeichnis aus, ergänzen Sie das darin enthaltene Programm so, dass es Ihren Namen mit ausgibt und committen Sie anschließend Ihre Änderungen.

Hinter Zeile 18 zum Beispiel für jeden Studierenden eine Zeile wie diese eingefügt:

```
System.out.println("Mustafa Müller");
```

b) Legen Sie nun ein neues Java-Projekt an und darin ein Programm, das alle durch 7 oder 13 teilbare Zahlen zwischen 10 und 40 (jeweils inklusive) ausgibt. Sichern Sie dieses in Ihrem SVN-Repository. Dazu können Sie den Befehl *Team->Share Project...* aus dem Kontextmenü des Projektes verwenden.

```
public class Zahlen {
  public static void main(String[] args) {

    for(int i = 10; i < 40; ++i) {
      if ((i % 7 == 0) || (i % 13 == 0)) {
        System.out.println(i);
      }
    }

  }
}
```

Aufgabe 2. *Dateioperationen mit SVN* (6 Punkte)

a) Erzeugen sie in Eclipse ein neues Projekt „Dateien“ und legen Sie darin einen Ordner „Nummer 1“ mit einer Datei „Datei 1.txt“an. Importieren Sie nun das Projekt ins SVN (*Team->Share project...*).

b) Fügen Sie in dem Ordner eine neue Datei „Datei 2.txt“ hinzu. Vergleichen Sie die Symbole der beiden Textdateien im *Package Explorer*. Was fällt auf? Welche Bedeutung haben die jeweiligen Symbole? Welche Schritte müssen Sie unternehmen, damit sich „Datei 2.txt“ im gleichen Zustand wie „Datei 1.txt“ befindet?

Datei 1.txt zeigt ein kleines Datenbanksymbol (= schon per SVN verwaltet), Datei 2.txt hingegen ein Fragezeichen (= noch nicht mit SVN verwaltet). Um auch Datei 2.txt zu verwalten, muss man auf der Datei den Kontextmenü-Befehl „Team->Add to version control…“ ausführen und dann das Project committen.

c) Benennen Sie nun „Datei 2.txt“ in „Datei 2a.txt“ um (Kontextmenü: *Refactor->Rename*). Was bemerken Sie? Was bedeutet dies für Dateioperationen bei SVN?

> Datei 2a.txt wird nun wieder als „Bearbeitet“ gekennzeichnet, d.h. SVN zeichnet Dateioperationen auf.

d) Öffnen Sie nun den Projektordner mit einem Dateiverwaltungsprogramm (z.B. Finder oder Windows Explorer)[1]. Benennen Sie nun hier, außerhalb von Eclipse, „Datei 1.txt“ in „XX.txt“ um. Wechseln Sie zu Eclipse und wählen Sie aus dem Kontextmenü des Projekts den Befehl „Refresh“ aus. Was bemerken Sie im Unterschied zu Aufgabenteil c) ? Welche Änderungen zeigt Ihnen die Sychronisations-Ansicht des Projekts (*Team->Synchronize*)?

> Die Datei XX.txt wird als neue (und nicht als geänderte) Datei angezeigt, die neu ins SVN System hinzugefügt werden muss. Die ursprüngliche Datei wird als gelöscht gekennzeichnet.

e) Committen Sie das Projekt. Benennen Sie nun außerhalb Eclipses den Ordner „Nummer 1“ in „Nummer 2“ um und aktualisieren Sie die Dateiansicht in Eclipse. Was bemerken Sie nun?

> Die Resourcen sind mit einem roten „Switched“-Pfeil gekennzeichnet. Dies liegt daran, dass das Umbenennen nicht von SVN aufgezeichnet werden konnte, aber ein versteckter „.svn“ Ordner im Verzeichnis weiterhin auf das ursprüngliche SVN Repository zeigt.

f) Wechseln Sie in die Synchronisations-Ansicht des Projekts. Was bemerken Sie nun? Was passiert bei einem Update? Was können Sie nun tun, um das Projekt zu „reparieren“?

> Für den Ordner wird ein Konflikt angezeigt, die Dateien sollen neu aus dem SVN geladen werden. Bei einem Update wird der ursprüngliche Ordner wieder hergestellt, der Ordner „Nummer 2“ befindet sich danach in einem Fehlerzustand. Man kann nun den Ordner „Nummer 2“ löschen oder darauf „Overwrite and Update“ ausführen. Dann zeigt der Ordner „Nummer 2“ auf das selbe SVN-Verzeichnis wie „Nummer 1“ und Dateiänderungen wirken sich jeweils wechselseitig aus.

Aufgabe 3. *Etiketten und Zweige* (15 Punkte)

Neben dem Hauptstamm eines mit SVN verwalteten Projektes (genannt „trunk“) gibt es noch die Konzepte der Etiketten („tags“) und Zweige („branches“).

a) Erklären Sie, welchen Zweck *tags* und *branches* erfüllen und wie diese in SVN angelegt werden.

> *Tags* beschreiben definierte Zustände, z.B. Releases, die benannt werden sollen. Mit *branches* werden nebenläufige Entwicklungszweige eines Projektes bezeichnet. In SVN erzeugte man beide durch das Erstellen einer Kopie von `trunk` mit sprechender Bezeichnung an einem speziell definierten Ort, zum Beispiel in den Verzeichnissen `tags` und `branches`.

Gibt es auf technischer Ebene im SVN Repository einen Unterschied zwischen *trunk*, *tags* und *branches*?

[1] Den Pfad des Projekts finden Sie in den Eigenschaften des Projektes bei „Resource“ unter „Location“.

Nein, beides sind Kopien von trunk. Per Definition werden in eine *tag*-Kopie jedoch keine Änderungen eingepflegt.

b) Legen Sie von Ihrem Projekt in Aufgabe 1b einen Tag an und benennen Sie diesen sinnvoll.

c) Erzeugen Sie einen Branch, checken Sie diesen aus und verändern Sie das verzweigte Projekt so, dass zusätzlich auch durch 5 teilbare Zahlen ausgegeben werden. Checken Sie das modifizierte Branch-Projekt wieder in das SVN Repository ein.

```
(…)
if ((i % 5 == 0) || (i % 7 == 0) || (i % 13 == 0 {
(…)
```

d) Ergänzen Sie das Programm im *trunk* so, dass durch 6 teilbare Zahlen übersprungen werden und committen Sie das Projekt. Fügen Sie nun den *branch* aus Aufgabenteil c wieder mit dem *trunk* zusammen („mergen“) und committen sie das Projekt erneut.

```
(…)
if (i % 6 == 0)
   continue;

if ((i % 5 == 0) || (i % 7 == 0) || (i % 13 == 0 {
(…)
```

e) Warum steigt beim Kopieren von Zweigen innerhalb des SVN Repositories der Speicherbedarf desselben – solange keine Modifikationen committet werden – nur minimal an?

Weil lediglich eine Liste der Revisions-Nummern der zur Kopie gehörigen Dateien erzeugt wird – die Kopie ist also nur „Virtuell“.

Aufgabe 4. *SVN Properties* (4 Punkte)

In Subversion lassen sich Dateien und Verzeichnisse um Meta-Informationen, die so genannten Eigenschaften („Properties“) ergänzen. Geben Sie für die folgenden Eigenschaften wieder jeweils deren Bedeutung und einen beispielhaften Wert an:

a) `svn:ignore` (für Verzeichnisse)

Liste der von SVN zu ignorierenden Dateinamen im Verzeichnis, z.B. `*.pdf`, `build`

b) `svn:mime-type` (für Dateien)

Bestimmt den Typ einer Datei und damit die Art und Weise, wie SVN die Datei behandelt (z.B. werden Binärdateien nicht zeilenweise zusammengefügt). Beispiele: `text/html`, `application/binary`

c) `svn:keywords` (für Dateien)

Liste der in der Datei automatisch zu ersetzenden Schlüsselwörter, z.B. `Id`

d) `svn:externals` (für Verzeichnisse)

Bildet ein anderes SVN-Projekt durch dessen absolute URL-Angabe in den Pfad dieses Projektes ein. Dies ermöglicht es, abhängige Projekte gemeinsam auszuchecken. Beispiel: `third-party/sounds http://sounds.red-bean.com/repos`

Weiterführende Informationen: http://svnbook.red-bean.com/en/1.0/ch07s02.html

Aufgabe 5. *Erste Schritte mit GIT* (12 Punkte)

Nach Kennenlernen von SVN, wollen wir uns mit Git beschäftigen und nachvollziehen, was daran anders / besser ist.

Installieren Sie dazu einen Git-Client auf ihrem System, und bearbeiten Sie dann die folgenden Aufgaben. Die Detailerläuterungen in der Aufgabenstellung beziehen sich auf die Arbeit mit SmartGit. Hinweise zur Installation von SmartGit erhalten Sie unter https://sewiki.iai.uni-bonn.de/teaching/lectures/se/2013/git

In dem Repository git://github.com/swt201112/uebungsblatt_1.git steht ein Text der aus drei Abschnitten besteht. Jeder von Ihnen soll *parallel zu den anderen Gruppenmitgliedern* auf seinem eigenen Rechner genau einen Abschnitt bearbeiten. Ordnen Sie sich selbst die Abschnitte zu, indem Sie ihre Vornamen alphabetisch sortieren. Der 1. Vorname bearbeitet den 1. Abschnitt, der 2. Vorname den zweiten Abschnitt, und so weiter. Wenn Sie mehr als 3 Teammitglieder sind, geht es zyklisch weiter, d.h. es gibt mehrere Leute, die den gleichen Abschnitt bearbeiten.

a) Holen Sie sich unser Repository auf ihren lokalen Rechner, indem sie ein Clone von git://github.com/swt201112/uebungsblatt_1.git durchführen.

b) Erstellen Sie nun für sich einen eigenen Branch, wechseln Sie zu dem neuen Branch und führen Sie darin folgende Änderungen durch:

   1. Korrigieren Sie die enthaltenen Rechtschreibfehler in der Datei *briand-kellogg.txt* entsprechend Ihrer vereinbarten Aufgabenteilung pro Abschnitt.
   2. Benennen Sie die Datei *briand-kellogg.txt* in *kellogg-briand.txt* um.
   3. Fügen Sie in der Datei *gedicht.txt* eine neue Strophe hinzu
   4. Committen Sie die Änderungen in ihr lokales Repository.

c) Übergeben Sie ihren lokalen Git-Ordner (gepackt) an alle ihre Teamkollegen (E-Mail, USB-Stick, ...)

d) Mergen Sie die Repositories ihrer Kollegen auf ihren eigenen Branch.
   - Dazu müssen Sie zuerst die Repositories unter Remotes/Manage Remotes als Remote Repository hinzufügen und einen Pull durchführen.
   - Eventuell auftretende Konflikte lösen Sie (in SmartGit) auf, indem sie auf die entsprechende Datei rechtsklicken und „Show Changes“ auswählen. In dem sich öffnenden Fenster manuell die Änderungen durchführen und speichern. Dann ein „Stage“ durchführen und anschließend wieder in Ihr lokales Repository committen.

e) Vereinbaren Sie, wer in ihrem Team als "Integrator" für die finale Version verantwortlich ist. Schicken Sie ihren Git-Ordner mit dem Zustand nach d) an ihren Integrator.

f) Der Integrator hat die Aufgabe alle ihre verschiedenen Branches zu einem zusammenzuführen und anschließend seinen Git-Ordner als ZIP-Datei gepackt als Gruppenabgabe in das SVN hochzuladen.

Lösungen zu dieser Aufgabe sind im Grunde direkt aus den Vorlesungsfolien des nachfolgend aufgeführten Foliensatzes herauszulesen.

```
Foliensatz:
02. Software Configuration Management (SCM)

Folien:
78 - 93
```