SVN, Maven Seminar

Aleksandar Tolev

7. Januar 2009

Inhaltsverzeichnis

1 Einleitung **2**

2 SVN/Subversion **3**
2.1 Definition und Geschichte des SVN 3
2.2 Arbeitsweise des SVN und Vorteile 4
2.3 Organisation des SVN-Repositories 6
2.4 Funktionsweise des SVN und Tools 7

3 Maven **9**
3.1 Die Philosophie Mavens 9
3.2 Die Phasen Mavens 11
3.3 Vergleich mvn, make und ant 12
3.4 Hudson continuous integration 13

4 Zusammenfassung **14**

1 Einleitung

Das folgende Seminar behandelt zwei wichtige Philosophien, die heutzutage in einem Projekt häufig Anwendung finden: SVN und Maven. Zum einen die Versionskontrolle SVN, mit deren Hilfe das Arbeiten an einer Datei innerhalb einer Gruppe möglich gemacht wird, sodass man immer die aktuellste Version der zu bearbeitenden Datei bereitgestellt bekommt. Mit Maven wird ein einheitliches Projektmanagement Tool bereitgestellt, welches den Entwicklern die Möglichkeit bietet, auf einem gemeinsamen Standard komplette Projektprozesse abzuarbeiten. Diese beiden Themen werden in Bezug auf das Studienprojekt 2008/2009 «DecidR »erötert und sollen zudem den «DecidR-Teammitgliedern »vermittelt werden, damit diese ein Verständnis für die beiden Technologien bekommen. Diese Techniken sollten nach Möglichkeit im Projekt angewendet werden. Das Seminar ist desweiteren ein Leistungsnachweis für den Kunden im Projekt und wird nach Abgabe und Präsentation benotet, die in die Gesamtnote des Projektes einfließt.

2 SVN/Subversion

2.1 Definition und Geschichte des SVN

Der Begriff Subversion bzw. Subversion ist klar definiert. Der Begriff bezieht sich zum einen auf die politisch-soziologische Deutung der Subversion, die eine Tätigkeit im Verborgenen, deren Ziel der Umsturz einer bestehenden Ordnung durch Unterwanderung und Untergrabung ist, definiert. Desweiteren bezieht sich der Begriff auf die sub version, also Unterversion oder der früheren Version. Das bedeutet Subversion dient der Versionskontrolle. Mit SVN können verschiedene Entwickler an Dateien arbeiten. Dabei wird immer die aktuellste Version bereitgestellt, sodass alle Entwickler auf dem gleichen Stand sind.
Subversion wurde seit Anfang 2000 bei CollabNet entwickelt und erreichte Februar 2004 seine stabile Version 1.0. Seitdem wurden immer neuere Versionen entwickelt, die letzte ist die Version 1.5 (Juni 2008).Folgend werden die Änderungen in den einzelnen Versionen aufgelistet:

Version 1.1 Repositories müssen nicht mehr in einer Berkley-Datenbank verwaltet werden, sondern man kann auch direkt das Dateisystem benutzen

Version 1.2 Die Sperrung von Dateien wurde eingeführt, was vor allem für binäre Dateien hilfreich ist

Version 1.3 Das Server-Logging, die Autorisierung, die Programmiersprachen-Anbindung, die Kommando-Option und die Leistung wurden verbessert

Version 1.4 svnsync bietet das Spiegeln von Projektarchiven an

Version 1.5 Merge-Tracking und das sparse checkout wurden eingeführt

2.2 Arbeitsweise des SVN und Vorteile

SVN arbeitet eigentlich wie das immer noch weitverbreitete CVS, welches viele Schwächen aufweist. SVN bietet eine Versionsverwaltung im Sinne einer einfachen Revisionszählung. Dabei werden die zu bearbeitenden Daten in einem Repository abgelegt. Falls Änderungen an den Dateien vorgenommen werden, werden lediglich die Unterschiede zwischen dem Repository und dem lokalen System übertragen.

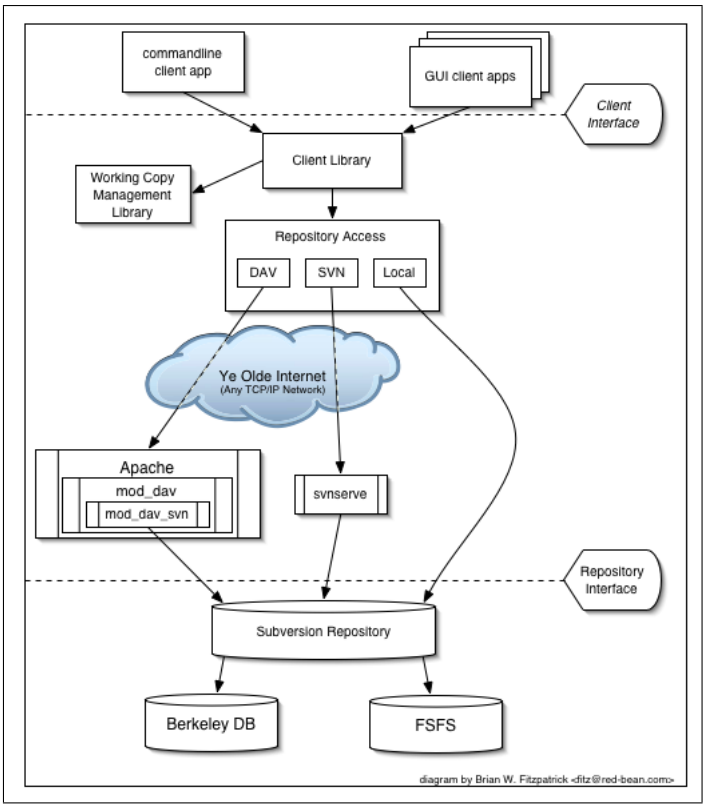

Abbildung 1: Subversion-Diagramm

Das Diagramm erläutert die Vorgehensweise des SVN in allen drei Schichten (User-Schicht, SVN-Schicht und Repository-Schicht). Über eine Applikation oder über die Kommandozeile gibt der User seine Interaktion an. Daraufhin greift SVN mittels des Befehls auf das Repository zu (über DAV, SVN oder lokal) und führt den Befehl auf dem Repository aus. Je nach Befehl werden Dateien hochgeladen, runtergeladen, hinzugefügt, gelöscht, gemerged etc.
SVN bietet zwei unterschiedliche Backends zur Verwaltung der Repositorys an. Das eine ist die *Berkley-Datenbank* und das andere ist das *fsfs-Backend*. In der folgenden Tabelle werden die Unterschiede und Gemeinsamkeiten aufgelistet:

Berkley-Datenbank	fsfs-Backend
Speichert das Repository in einer Datenbank	Speichert das Repository in einer einzigen Datei
Transaktionen werden in der Datenbank gespeichert	Transaktionen werden als Unterordner gespeichert
Die Größe und der Zugriff spielen keine Rolle	Die Größe ist kleiner und der Zugriff ist nicht unbegrenzt
Benutzt einen $O(n^2)$-Algorithmus, um den kompletten Ordner zu überschreiben	Benutzt einen O(n) -Algorithmus, um die Dateien an das Repository ranzuhängen
Auscheken des letzten Baumes ist schnell	Auschecken des letzten Baumes ist etwas langsamer
Beim Crash bleibt die DB unbrauchbar bis sie repariert wird	Beim Crash ist nicht das komplette Repository betroffen, sondern evtl. nur die Transaktion
Repository-Backup funktioniert zur Laufzeit	Repository-Backup funktioniert zur Laufzeit
Repository kann nicht auf andere OS kopiert werden	Repository kann auf andere OS portiert werden

Im Gegensatz zu CVS bietet SVN einige Merkmale, die das Arbeiten sehr viel einfacher gestalten. In der folgenden Liste, sind die Vorteile gegenüber CVS aufgelistet:

- SVN versioniert oder revisioniert grundsätzlich das gesamte Projektarchiv und damit jeweils die gesamte Verzeichnisstruktur, während CVS auf der unabhängigen Versionierung jedes einzelnen Inhalts beruht
- Mit Subversion ist es möglich, Dateien oder Verzeichnisse zu verschieben oder umzubenennen, ohne die Versionsgeschichte zu verlieren
- Änderungen ("commits") sind in Subversion atomar, das heißt, eine Änderung wird entweder ganz oder gar nicht ins Repository gespeichert. Verbindungsabbrüche und mehrere gleichzeitige "commits"können somit nicht zu inkonsistenten Zuständen führen

2.3 Organisation des SVN-Repositories

Die Sturkturierung der Ordner ist wohl das wesentliche Hauptmerkmal des SVN. SVN kennt nur die Idee der Kopie und dementsprechend wird auch zur solch einer Ordnerstruktur geraten. Das bedeutet, man besitzt eine Hauptentwicklungslinie die den Stamm des kompletten Projektes bildet. Im Falle des Studienprojektes «DecidR »ist trunk der Hauptornder, indem das Hauptprojekt abgespeichert wird. Das es dabei sinnlos ist, alle Dateien nur in dem trunk Ordner zu legen ist eine weitere Untergliederung sehr sinnvoll. Es wurden die Ordner docs, prototype, seminars und src angelegt. Im Unterordner docs befinden sich alle Dokumente die in diesem Projekt erstellt werden (Angebot, Anforderungsanalyse, Spezifikation, Entwurf, Test, Handbuch usw.). Für jedes weitere Dokument wurd ein weiterer Unterordner angelegt, indem dann die kompletten Dateien abgelegt werden. Der Unterordner prototype dient zur Ablage der Dateien für den Prototypen. Auch dieser enthält Unterordner, die sich speziell auf das Subprojekt beziehen. Der Ordner seminars bietet den Teammitgliedern die Möglichkeit ihr eigenes Seminar per Versionskontrolle zu erstellen. Jedes Teammitglied besitzt einen eigenen Ordner, den er selber weiter gestalten kann, wie er möchte. Und im Unterordner src wird der Hauptcode abgelegt, der während der Implementierungsphase ensteht, abgelegt. Zudem gibt es noch den Ordner tags. Dieser Ordner ist dazu da, um alle Versionen festzuhalten, die als fertig erachtet werden. Das bedeutet, falls der erste Prototyp lauffähig ist, wird die Version dort abgelegt, damit die Entwickler zu einem späteren Zeitpunkt genau wissen auf welche Version sie sich berufen müssen, falls sie etwas am fertig Prototypen ändern möchten. Nun kann man diskutieren inwieweit das sinnvoll ist, da SVN ja die Versionverwaltung anbietet und man dadurch auch auf frühere Versionen zugreifen kann. Wir haben es als sinnvoll erachtet und dementsprechend den Ordner angelegt. Letztlich bleibt noch der Ordner branch. In diesem Ordner werden nach der Philosophie des SVN Kopien des Stammordners trunk angelegt, um, wenn evtl. nötig, andere Entwicklungspfade einzuschlagen. Der Ordner ist somit ein Zweig der Hauptentwicklungslinie. Man kann dann andere Entwicklungpfade gehen, ohne den Stamm dabei zuverlieren.
Diese Ordnerstruktur wird als Standard angesehen und von allen SVN Betreibern empfohlen. Letzten Endes bleibt es dem Administrator überlassen, wie er das SVN strukturiert.
Subversion kann im Gegensatz zu CVS, auch mit Binärdateien umgehen. Bei Binärdateien werden lediglich die Änderungen übertragen. Jedoch lassen sich die Binärdateien nicht einfach so zusammenführen. Daher muss die zu bearbeitende Binärdatei gesperrt werden, bevor sie bearbeitet werden kann. Wenn die Bearbeitung fertig ist, wird sie wieder freigegeben und die Änderung wird beim nächsten Update übertragen.

2.4 Funktionsweise des SVN und Tools

Im vorigen Unterkapitel wurde erläutert, wie das SVN funktioniert. In diesem Unterkapitel wird erklärt was für Funktionalitäten SVN bietet, bzw. wie man mit SVN arbeitet. Anschließend werden Tools für die Arbeit mit SVN aufgelistet. Bei den Befehlen bezieht sich die angegeben Adresse auf das SVN-Repository vom Studienprojekt «DecidR ».
Nachdem das SVN Repository auf einem Server angelegt und die Ordnerstruktur gewählt wurde, muss der Entwickler die gleiche Ordnerstruktur auf sein OS herstellen. Dazu bietet SVN einen Befehl an:

```
svn checkout http://decidr.googlecode.com/svn/trunk/ decidr
```

Dieser Befehl berechtigt den Entwickler nur das Lesen der Ordnerstruktur. Falls man selber Dateien hochladen oder Änderungen vornehmen möchte, muss das Checkout verschlüsselt passieren:

```
svn checkout https://decidr.googlecode.com/svn/trunk/
decidr --username username
```

Man muss dabei seinen Benutzernamen und sein Passwort eingeben, erst dann erfolgt der Checkout in den Ordner, den man nach der Repository-Adresse angegeben hat. Beim ersten checkout wird in dem Ordner eine .svn Datei erstellt, in der die kompletten Daten des SVN-Repositorys stehen.
Sämtliche Updates auf dem Repository kann mit dem Update-Befehl auf sein OS holen:

```
svn update decidr
```

Dabei reicht es, wenn man den Ordner angibt der erneuert werden soll. Dieser muss aber die .svn Datei erhalten.
Falls man neue Dateien in das Repository laden möchte, muss man zunächst die Dateien vormerken. Dies geschieht mit:

```
svn add file
```

Daraufhin muss dem Repository noch mitgeteilt werden, dass die markierten Dateien nun hochgeladen und ins Repository aufgenommen werden sollen:

```
svn commit decidr
```

Beim commit ist es üblich, dass man noch Kommentare mitschreibt, zum besseren Verständnis was sich geändert hat. Das erreicht man mit dem Parameter -m «Kommentar ».

Tools für SVN

- rapidSVN [1]
- subclipse [2]
- subcommander [3]
- TortoiseSVN [4]
- versions [5]
- statsvn [6]
- smartSVN [7]

3 Maven

Maven kommt aus dem Jüdischen (*meyvn*) bzw. aus dem Hebräischem (*mevin*) und bedeutet soviel wie Experte. Die genaue englische Übersetzung lautet accumulator of knowledge, quasi ein Speicher für Wissen.
Maven wurde als Ansatz im *Apache Jakarta Turbine Project* [1] benutzt, um den Build-Prozess zu vereinfachen. Es gab einige verschiedene Projekte mit ihren eigenen Ant-Buildfiles, die alle unterschiedlich waren und die Jar-Dateien wurden alle ins CVS geladen.
Daraus resultierte der Wunsch eines einheitlichen automatisierten Verfahrens, um den kompletten Build-Prozess zu standardisieren. Dabei setzt sich Maven mit folgenden Zielen auseinander:

- Vereinfachung des Build-Prozesses
- Bereitstellung eines einheitlichen Build-Systems
- Bereitstellung von Qualitätsinformationen bezüglich des Projektes
- Bereitstellung von Richtlininen zur besten Entwicklung
- Transparente Migration von neuen Features (Pklug-In)

Außerdem versucht Maven das Prinzip *Convention over Configuration*[2] für den gesamten Software-Lebenszyklus zu verfolgen.
Die erste Version von Maven war noch stark an Ant angelehnt, sodass sich diese beiden Build-Methoden ähnelten. Die zweite Version verfolgte grundlegende neue Ansätze und ist nicht mehr mit der vorherigen Version kompatibel.

3.1 Die Philosophie Mavens

pom.xml

Beim Erstellen eins Maven Projektes wird ein einheitliches Projektverzeichnis und eine einheitliche Projektstruktur verwendet. Maven erstellt eine XML-Datei, die sogenannte *pom.xml*. In dieser stehen die Grunddaten für das Projekt. Zum Beispiel stehen in der Datei, der Name des Projektes, die Version, die Art des Paketes, das später erstellt werden soll, die URL, die sich auf das Projekt bezieht und die Abhängigkeiten, die das Projekt mit sich bringt und aufgelöst werden sollen, wenn das Projekt erstellt wird.

settings.xml

Zusätzlich wird eine weitere Konfigurationsverwaltung erstellt, die *settings.xml*. In ihr werden die Zugangsdaten für den Repository-Server und Deploy-Server gespeichert. Desweiteren werden die Einbindung von Plugins- oder Bibliotheken-Repositorys oder projektübergreifende Parameter eingestellt. Sowohl in der settings.xml als auch in der pom.xml gibt es sogennante Profile, die ein weiterer Mechanismus zur Änderung der Einstellungen sind. In ihr können abweichende Parameter gesetzt werden, wie z.B. für Filterdateien oder Repositorys.

.m2-Ordner

Wie schon erwähnt werden bei der ersten Verwendung diese beiden XML-Dateien

[1] Turbine ist ein Servlet basiertes Framework für erfahrene Java-Entwickler, um schnell Web-Applikationen zu erstellen

[2] Das Ziel ist, die Zahl der Entscheidungen, die ein Entwickler trifft, zu verringern und dabei Einfachheit zu erreichen, ohne die Felxibiltät zu beeinflussen

erstellt. Zusätzlich wird noch ein weiterer Ordner erstellt. Dieser heißt .m2 Ordner und in ihm befindet sich das lokale Repository, welches mit den Dateien des entfernten Maven-Repositorys gefüllt wird. Somit ist gewährleistet, dass bei einem erneuten Nutzen von Maven nicht alles aus dem entfernten Repository runtergeladen wird, sondern Maven greift nur noch auf das lokale zu, sofern die benötigten Pakete dort schon vorhanden sind. Wenn nicht, werden diese automatisch runtergeladen.

Zentrales Repository

Ein weiterer Grundgedanke besteht darin, sämtliche Bibliotheken und Plugins an einem zentralen Ort zu verwalten. Dies vereinfacht bei vielen Projekten den Verwaltungsaufwand. Diese zentrale Repository bietet die Apache Group an. Man kann auch eigene Repositorys erstellen, der Vorteil besteht darin, dass man keine Internetverbindung benötigt und eine firmenübergreifende Bibliothek erstellen kann. Der Vorteil des zentralen Repositorys liegt darin, dass Abhängigkeiten bereits abgelegt sind, und nicht manuell mit eingebaut werden müssen.

IDE

Es gibt für Eclipse und NetBeans Plugins, um Maven in den IDEs zu integrieren und dort auch zu nutzen.

Man sieht also, dass mit wenigen Konfigurationseinstellungen der Maven-Prozess verändert werden kann. Aber diese Einstellungen ändern nichts am automatisierten Vorgehen von Maven, dieser bleibt immer gleich. Das ist ein großer Vorteil. Den Maven baut seine Projekte durch das Project Object Model (POM) und somit ist ein einheitliches Build-System gewährleistet, was es wiederrum möglich macht alle Maven Projekte zu verstehen. Dadurch spart man Zeit bei der Einarbeitung von anderen Projekten, da man die Bauweise kennt.

Maven basiert auf einer Plugin-Architektur, die es erlaubt verschiedene Plungins für verschiedene Anwendungen auf das Projekt anzuwenden, ohne diese explizit installieren zu müssen. Daher ist die Migration von Features sehr einfach gehalten und einfach möglich.

Was Maven nicht ist!

Zum Schluss noch einige Dinge darüber was Maven nicht ist. Maven ist keine Erweiterung des Build-Tools Ant und ebenfalls nicht nur ausschließlich ein Dokumentations-Tool. Maven stellt die besten Möglichkeiten zur Verfügung. Es kann durchaus sein, dass es manchmal nicht möglich ist mit Maven ein Projekt zu bauen, weil die Struktur des Projektes so exotisch ist, dass eine Standardisierung gar nicht möglich ist. Dann sollte man ,in diesem Fall, Maven lieber aufgeben und versuchen das Projekt auf eine andere Weise zu builden.

Beim kompletten Build-Prozess durchläuft Maven verschiedene Phasen, in denen jeweils ein bestimmter Teil des Software-Lebenszyklus durchgearbeitet wird. Diese werden im nächsten Abschnitt beschrieben.

3.2 Die Phasen Mavens

Validierungsphase	Hier wird die pom.xml auf ihre Struktur überprüft, ob diese korrekt ist. Erst wenn die korrekt ist, kann der Prozess weitergeführt werden
Kompilierungsphase	Hier wird das Projekt kompiliert. Beim ersten Kompilieren, werden alle nötigen Plugins heruntergeladen. Danach greift Maven auf das lokale System zu, falls es Plugins benötigt und sucht dort nach ihnen. Maven kompiliert die Klassen, die im Standard-Verzeichnis liegen. Die werden dort beim Erstellen des Projektes hineingelegt.
Testphase	Hier werden die generierten Testklassen kompiliert und ausgeführt. Dabei werden zusätzliche Plugins für das Kompilieren der Testklassen heruntergeladen. Bevor die Testklassen ausgeführt werden, wird der Maincode kompiliert.
Paketierungsphase	In dieser Phase wird eine jar-Datei erstellt. In der pom.xml kann man angeben, was in dieser Phase erstellt werden soll. Die jar-Datei wird in einem standardisiertem Verzeichnis abgelegt.
Integrations-Test	Hier wird durch ein Plugin das fertig Paket in eine Umgebung integriert und getestet.
Verifizierungsphase	Hier wird das Paket überprüft, ob es bestimmte Dateien/Ordner erhält und prüft den Inhalt auf ihre Struktur.
Installationsphase	In dieser Phase wird das Paket, das im lokalen Maven-Repository liegt, installiert.
Deployment	Ein Plugin für Maven sorgt dafür, dass das erstellte Paket auf einem entfernten Repository gelegt wird, sodass andere Entwickler das Paket nutzen können. Alle nötigen Informationen werden aus der pom.xml entnommen, sodass ein problemloses Deployment stattfinden kann.
Site	Nun wird eine Seite erstellt, mit allen Informationen bezüglich des Projektes, sodass alle Informationen an einem Punkt sind.
Clean	Clean löscht den Zielordner mit all seinen Builddaten.

3.3 Vergleich mvn, make und ant

Maven ist das zweite Buildprojekt von Apache. Das erste Buildprojekt war *ant*. Ant ist wohl das heutzutage weit verbreiteste Build-Tool. Es existiert noch ein weiteres Build-Tool und zwar *make*. *Make* ist Teil des POSIX-Standards, dessen gegenwärtige Bezeichnung IEEE Std 1003.1, 2004 Edition lautet.

Apache-Ant

Apache Ant ist ein Java-basiertes Werkzeugt zur automatischen Erzeugung von Programmen aus Quelltext. Ant ist ein Akronym und steht für Another Neat Tool. Es wurde 1999 von James Duncan Davidson entwickelt, der ein Werkzeug wie make nur für Java benötigte. Ant wird durch eine einzige XML-Datei gesteuert, der sogenannten *build.xml*. In dieser Build-Datei wird ein Project ermittelt, welches das Wurzelelement darstellt. Desweiteren besitzt die build.xml sogenannte *Targets*. Diese sind vergleichbar mit Funktionen in Programmiersprachen und können von außen über die Kommandozeile aufgerufen werden. Abhängigkeiten, die in den Targets definiert werden, löst Ant auf und dient zum Bauen des Programmes. Da es sich um eine XML-Datei handelt ist diese nicht von Tabulatorzeichen oder sonstiges abhängig. Dies ist eine Verbesserung gegenüber dem alten Build-Tool make.

make

Make liest ein sogenanntes *Makefile*, in dem der Übersetzungsprozess von Programmen formalisiert erfasst ist. Diese Formalisierung beschreibt, welche Quelltextdateien der Compiler zu welchen Objektdateien verarbeitet, und welche Objektdateien vom Linker dann zu Programmbibliotheken oder ausführbaren Programmen verbunden werden. Alle Schritte erfolgen unter Beachtung der Abhängigkeiten, die möglicherweise durch die Dateiorganisation gegeben sein können. Bei der Abarbeitung wird eine Umwandlung von einer Quelldatei nur dann vorgenommen, wenn die Quelldatei neuer als die Objektdatei ist. Der Vorteil dabei ist, dass bei kleinen Veränderungen in großen Projekten nicht die komplette Kompilation durchgeführt werden muss. Jedoch müssen alle Abhängigkeiten in der Makefile beschrieben werden, was bei sehr großen Projekten nicht immer leicht zu realisieren ist.

Vor- und Nachteile von mvn, ant und make

Der klare Vorteil bei Ant und Make ist die Vielzahl an Literatur, die es mittlerweile gibt und die praktische Erfahrung, da diese beiden Build-Tools am weitesten verbreitet sind. Außerdem ist die Einarbeitung in Ant einfacher und es ist auch leichter zu handhaben. Die Einarbeitung in Maven fällt wesentlicher größer aus. Dafür bietet Maven eine einheitliche Projektstruktur und eine zentrale Verwaltung von Bibliotheken durch das Repository. Zudem ist es leicht erweiterbar, was für ant und make nicht gilt. Bei make spielen die Abhängigkeiten eine große Rolle und können bei großen Projekten zu Problemen führen, um die sich Maven bei seinen Projekten nicht kümmern muss. Einen weiteren Zusatz bietet das site von Maven, in der Informationen zum Projekt bereitgestellt werden. Maven und ant sind XML-basiert im Gegensatz zu make, welches einem Shellskript ähnelt.

3.4 Hudson continuous integration

Hudson contiuous integration ist ein erweiterbares, webbasiertes Tool zur kontinuierlichen Integration in agilen Softwareprojekten. Kontinuierliche Integration bedeutet eine fortlaufende, permanente Integration der den Prozess des vollständigen Neubildens und Testens einer Anwendung beschreibt. Dabei spielt das agile Softwareprojekt eine bedeutende Rolle. Denn durch die Agilität der Software wird die permanente Integration möglich. Das Extreme Programming ist ein Beispiel für solch ein agiles Softwareprojekt. Sobald Änderungen in der Anwendung vorgenommen werden, wird die komplette Anwendung neu gebaut und automatisch getestet. Falls dieser Test erfolgreich ist, wird die Anwendung in die nächste Stufe gereicht, falls nicht, findet ein Rollback statt und die Entwickler werden aufgefordert die Anwendung zu verbessern. Und genau das übernimmt Hudson, in automatisierter Form.

Funktionalitäten

Hudson wurde in erster Linie von Kohsuke Kawaguchi, Mitarbeiter von Sun Microsystems, entwickelt. Bis auf die Icons steht das komplette Programm unter der MIT-Lizenz. Hudson ist in Java geschrieben und läuft auf einem beliebigen Servlet-Container. Es werden sämtliche gängigen Build-Tools verwendet, wie Apache-Ant oder Apache-Maven. Zusätzlich bietet Hudson noch die Möglichkeit der Versionsverwaltung (CVS oder Subversion). Desweiteren bietet Hudson die Möglichkeit Änderungen per Email zu verschicken und so die betreffenden Entwickler zu informieren.

Vorteile

Dabei steht die Einfachheit an erster Stelle. Gerade in Bezug auf Maven bietet Hudson eine sehr einfache und praktikable Möglichkeit. Es ist möglich ein Maven Projekt zu erstellen, sodass Hudson es überwacht und Änderungen zur sofortigen Neubildung und Testen der Anwendung führt. Durch die kontinuierliche Integration, die Hudson bietet, wird die Programmierung vorteilhafter. Dabei werden

- Fehler laufend entdeckt und behoben und nicht erst vor dem Meilenstein
- frühe Warnungen bei nicht passenden Bestandteilen ausgegeben
- Fehler bei Unit-Tests sofort entdeckt
- lauffähige Versionen für Test-, Demo- und Vertriebszwecke angeboten
- die Entwickler gezwungen verwantwortungsvoller mit ihren Tätigkeiten umzugehen, um Fehler zu vermeiden

4 Zusammenfassung

Im der Ausarbeitung habe ich zwei gängige Tools beschrieben, die heutzutage als Standard gelten, wenn man größere Projekte entwickeln möchte.
Subversion ist nicht mehr vom Projektmanagement weg zu denken. Es bietet eine stabile Grundlage zur Sicherung der Dateien und zur Verwaltung der Dateien, sodass eine große Softwaregruppe mit bis zu zehn Entwicklern gleichzeitig an einem Projekt arbeiten kann, ohne dass es zu Konflikten kommt. Und durch das ausgeprägte Loggen kann man immer nachvollziehen, welche Änderunge vorgenommen wurden. Zum anderen dient Subversion ebenfalls als Nachweis. Durch weitere Programme kann man sich die kompletten svn-Daten auflisten lassen und kann überprüfen, welcher Entwickler wieviel hochgeladen hat. Die müssen nicht ausschlaggebend für eine Bewertung des Entwicklers sein, aber man kann sie in die Bewertung mit einbeziehen. Man sieht also, dass Subversion mehr zu bieten hat als nur eine Versionsverwaltung.
Maven erleichtert einige Aufgaben eines Entwicklers oder einer Entwicklergruppe. Der standardisierte Software-Lebenszyklus wird automatisiert und unterschiedliche Strukturen werden aufgehoben, sodass eine einheitliche Anwendung entsteht, die später ohne große Probleme in anderen Projekten integriert werden kann. Dabei werden frühzeitig Fehler entdeckt, die Entwickler vor großen Problemen stellen könnten. In Kombination mit Hudson, der permanent bei neuen Änderungen die Anwendung neu baut und testet, ist Maven ein weiteres Tool für das Projektmanagement, mit dem Entwickler sehr viel Zeit sparen können und diese Zeit in Phasen investieren können, die der Planung des Projektes dienen (Projektplan, Spezifikation, Entwurf). Das gewährleistet eine bessere Planung mit hoher Qualität, wenigen Fehlern und einer stabilen Software.
In Bezug auf das Studienprojekt «DecidR »wird Subversion schon benutzt. Ob sich Maven und Hudson mitintegrieren lassen wird sich entscheiden. Meine persönlicher Wunsch wäre es, diese Tools zu benutzen.

Literatur

[1] http://rapidsvn.tigris.org/.

[2] http://subclipse.tigris.org.

[3] http://subcommander.tigris.org/.

[4] http://tortoisesvn.tigris.org/.

[5] http://versionsapp.com/.

[6] http://www.statsvn.org/.

[7] http://www.syntevo.com/smartsvn/index.html.