Die Entwicklungswerkzeuge SVN und Maven

Aleksandar Tolev

Institute of Architecture of Application Systems, University of Stuttgart, Germany
Universitätsstrasse 38, 70569 Stuttgart, Germany
tolevar@studi.informatik.uni-stuttgart.de

Zusammenfassung In dieser Ausarbeitung werden die zwei Werkzeuge, Subversion und Maven, behandelt. Sie werden in einem Softwareprojekt häufig zur Versionierung und zur vereinfachten Kompilierung eingesetzt.

1 Einleitung

Heutzutage sind Softwareprojekte im größeren Umfang keine Seltenheit mehr. Dabei ist es wichtig einen Überblick über die Dateien und ihre Änderungen zu haben. Das übernimmt die Versionskontrolle Subversion. Mit ihrer Hilfe ist die Möglichkeit geboten stets mit der aktuellsten Version einer Datei zu arbeiten. Durch Subversion können mehrere Projektlinien des gleichen Hauptprojektes gleichzeitig verfolgt werden. Dabei entstehen keine Komplikationen innerhalb des Projektes. Im Fehlerfall können vorgenommene Änderungen zurückgenommen werden, indem man zu einer vorherigen Version zurückkehrt.

Durch das Arbeiten in großen Teams ist nicht immer eine gemeinsame Projektstruktur gegeben, da unterschiedliche Entwickler unterschiedliche Strukturen erstellen. Dadurch kann es zu nicht einheitlichen Ordnerstrukturen kommen, was zu unterschiedlichen Buidlfiles führt. Mit Maven wird ein einheitliches Build-Tool bereitgestellt, welches den Entwicklern die Möglichkeit bietet, auf einem gemeinsamen Standard die Implementierung abzuarbeiten.

Diese beiden Themen werden in Bezug auf das Studienprojekt 2008/2009 „DecidR“ vorgestellt und sollen zudem den „DecidR-Teammitgliedern“ vermittelt werden, damit diese ein Verständnis für die beiden Werkzeuge Subversion und Maven bekommen. Diese Werkzeuge sollten nach Möglichkeit im Projekt angewendet werden. In Abschnitt 2 wird das Werkzeug „Subversion“ vorgestellt. Daran schließt sich Abschnitt 3 an, in dem Maven beschrieben wird. Den Abschluss der Arbeit bildet die Zusammenfassung in Abschnitt 4.

2 Subversion

2.1 Definition und Geschichte des Subversion

Definition von SVN

Der Begriff „Subversion“ selbst ist klar definiert. Der Begriff bezieht sich zum einen auf die politisch-soziologische Deutung [1] der Subversion. Diese definiert eine Tätigkeit im Verborgenen, deren Ziel der Umsturz einer bestehenden Ordnung durch Unterwanderung und Untergrabung ist. Im Bezug auf Subversion

(Abkürzung: SVN) gibt es insofern einen Zusammenhang zu dieser Definition, dass die bestehende Version durch eine neue Version ersetzt wird und dabei die eigentliche Tätigkeit im Hintergrund bleibt. 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.

Geschichte des SVN

Subversion wurde seit Anfang 2000 bei CollabNet entwickelt und erreichte Februar 2004 seine stabile Version 1.0. Seitdem wurden immer neuere Versionen entwickelt, die derzeit aktuellste Version ist 1.5 vom Juni 2008 [2]. Folgend werden die wichtigsten Änderungen in den einzelnen Versionen aufgeführt:[2]

Version 1.1 Repositories müssen nicht mehr in einer Berkley-Datenbank [2] verwaltet werden, sondern man kann auch direkt das Dateisystem benutzen

Version 1.2 Die Sperrung von Dateien wurde eingeführt

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. Das sparse checkout ermöglicht das Auschecken von einzelnen Dateien oder Ordnern

2.2 Arbeitsweise des SVN und Vorteile

Arbeitsweise SVN

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.

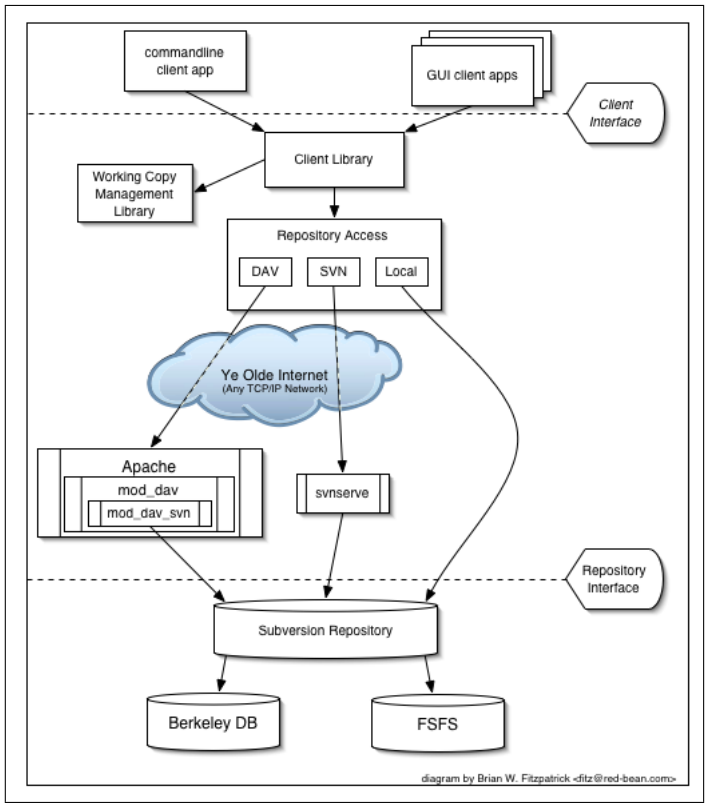

Abbildung 1. Subversion-Infrastruktur [2]

Infrastruktur

Das in Abbildung 1 gezeigte Bild erläutert die Vorgehensweise des SVN in der User-Schicht, SVN-Schicht und Repository-Schicht. Die User-Schicht beschreibt die Interaktion des Users mit dem SVN-Repository mit Hilfe einer Applikation oder der Kommandozeile. Daraufhin greift SVN in der SVN-Schicht mittels des Befehls auf das Repository zu (über DAV, SVN oder lokal) und führt den Befehl auf dem Repository aus (Repository-Schicht). Je nach Befehl werden Dateien hochgeladen, heruntergeladen, hinzugefügt, gelöscht, gemerged etc.

SVN versioniert die kompletten Dateien und Verzeichnisse im Repository. Außerdem ist SVN in der Lage, Kopien von Dateien zu verwalten, die die gleiche Geschichte teilen. Dadurch ist es möglich verschiedene Projektlinien zu verfolgen, ohne dabei die Hauptlinie zu beschädigen. Üblicherweise wird der Hauptzweig in den Ordner `trunk` gelegt und Abzweige davon in den Ordner `branches` (vgl. Abschnitt 2.3).

Backends für SVN

SVN bietet zwei unterschiedliche Backends zur Verwaltung der Repositorys an. Das eine ist die *Berkley-Datenbank* und das andere ist das *Filesystem-Backend*. In Tabelle 2.2 werden die Unterschiede und Gemeinsamkeiten gezeigt:

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

Tabelle 1. Vergleich Berkley-Datenbank-Backend und fsfs-Backend [2]

Vorteile

Neben SVN gibt es das Concurrent Version System [3] (kurz: CVS) zur Versionskontrolle. Dieses speichert alle Daten an einer zentralen Stelle, dem Repository. Im Gegensatz zu CVS bietet SVN einige Merkmale, die das Arbeiten 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. Dies bedeutet,dass eine Änderung entweder ganz oder gar nicht ins Repository gespeichert wird. Verbindungsabbrüche und mehrere gleichzeitige commits können somit nicht zu inkonsistenten Zuständen führen
- Subversion kann im Gegensatz zu CVS, einfacher mit Binärdateien umgehen. Bei Binärdateien werden die Differenzen übertragen. Daher wird eine Binärdatei im Laufe eines Projektes immer größer. Jedoch lassen sich die Binärdateien nicht einfach so mergen. 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.

2.3 Organisation des SVN-Repositories

Ordnerstruktur

Die Strukturierung der Ordner ist wohl das wesentliche Hauptmerkmal des SVN. SVN arbeitet mit der Idee der Kopie. Das bedeutet, man besitzt eine Hauptentwicklungslinie die den Stamm des kompletten Projektes bildet. Im Falle des Studienprojektes „DecidR“ (vgl. Abbildung 2) ist `trunk` der Hauptordner, indem das Hauptprojekt abgespeichert wird. Da es dabei sinnlos ist, alle Dateien nur in dem `trunk` Ordner zu legen, ist eine weitere Untergliederung 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 wurde 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.

Zudem gibt es noch den Ordner `tags`. Dieser Ordner dient der Markierung von gewissen Versionsständen. Hier werden alle Versionen abgelegt, die als stabil erachtet werden oder als Review-Version dienen. 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 fertigen Prototypen ändern möchten.

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 einschlagen, ohne die Hauptentwicklungslinie zu beschädigen.

Diese Ordnerstruktur wird als Standard angesehen und von allen SVN Betreibern empfohlen. Letzten Endes bleibt es dem Versionsbeauftragtem überlassen, wie er das SVN strukturiert. Der Versionsbeauftragte ist die Person im Projekt, die sich um die Ordnerstruktur im SVN kümmert.

2.4 Funktionsweise des SVN und Tools

Funktionsweise des SVN

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 angegebene Adresse auf das SVN-Repository vom Studienprojekt „DecidR“.

Befehle

Nachdem das SVN-Repository auf einem Server angelegt und die Ordnerstruktur gewählt wurde, muss der Entwickler die gleiche Ordnerstruktur auf seinem Betriebssystem (OS) herstellen. Dazu bietet SVN einen Befehl an:

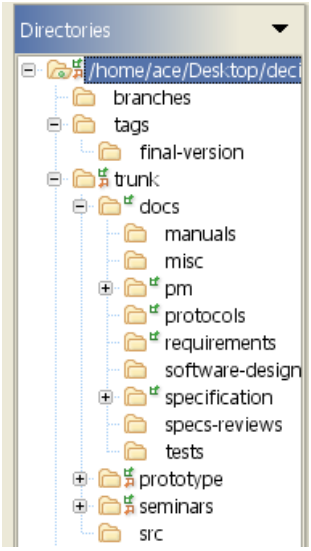

Abbildung 2. Ordnerstrukur des Studienprojektes „DecidR“

```
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 geschehen:

```
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 man 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 enthalten.

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 der Änderungen. Das realisiert man mit dem Parameter -m „kommentar“.

Um Konflikte zu vermeiden, falls zwei Entwickler gleichzeitig an einer Datei arbeiten, kann man mit Hilfe von SVN Dateien sperren lassen:

```
svn lock filename
```

Diese müssen nach Beenden der Änderung wieder entsperrt werden:

```
svn unlock filename
```

2.5 Tools für SVN

Tools

Für SVN gibt es eine Reihe von Tools. Hier werden welche für MacOS, Windows, Linux und plattformunabhängige Systeme vorgestellt. Plattformunabhängige Tools sind Tools, die auf jeder Plattform benutzt werden können. Im Studienprojekt benutzen wir das plattformunabhängige Tool smartSVN. Wir haben dieses Tool ausgewählt, damit wir ein einheitliches Tool besitzen, da unsere Teammitglieder mit Windows, Linux und MacOS arbeiten.

Hier einige plattformunabhängige Tools:

- rapidSVN [4]
- subcommander [5]
- smartSVN [6]

Ein Tool für Windows, welches nicht auf einer Applikation beruht, sondern in Form des Contextmenüs alle Funktionen verfügbar macht:

- TortoiseSVN [7]

Ein Tool für MacOS:

- versions [8]

Ein Tool für Eclipse, welches wir benutzen, um den Source-Code direkt in Eclipse auschecken, bzw. updaten und committen zu können:

- subclipse [9]

3 Maven

Definition

Der Begriff „Maven“ selbst stammt aus dem Jüdischen (*meyvn*) bzw. aus dem Hebräischem (*mevin*) und bedeutet soviel wie „Experte“. Die genaue englische Übersetzung lautet „accumulator of knowledge“ [10], quasi ein Speicher für Wissen.

Maven wurde als Ansatz im *Apache Jakarta Turbine Project* benutzt, um den Build-Prozess zu vereinfachen. Turbine ist ein Servlet basiertes Framework für erfahrene Java-Entwickler, um schnell Web-Applikationen zu erstellen. 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 (mittels Plugins)

Außerdem versucht Maven das Prinzip *Convention over Configuration* [11] für den gesamten Software-Lebenszyklus zu verfolgen. Dabei ist das Ziel, die Zahl der Entscheidungen, die ein Entwickler trifft, zu verringern und dabei Einfachheit zu erreichen, ohne die Flexibiltät zu beeinflussen.

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 Grundsätze und Eigenschaften Mavens

Zentrales Repository

Ein Grundgedanke besteht darin, sämtliche Bibliotheken und Plugins an einem zentralen Ort zu verwalten. Dies vereinfacht bei vielen Projekten den Verwaltungsaufwand. Dieses 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.

pom.xml

Beim Erstellen eines Maven Projektes kann der Benutzer die Ordnernamen und die Ordnertiefe genau festlegen. Dabei verfolgt Maven eine einheitliche Struktur. Java Klassen werden in der niedrigsten Ordnertiefe abgelegt. Maven bezieht sich beim Builden immer auf die unterste Stufe der Ordnerstruktur, sodass eine Automatisierung ermöglicht wird. Maven erstellt eine XML-Datei, die sogenannte `pom.xml`. In dieser stehen die Grunddaten für das Projekt, welche aus dem Project Object Model (POM) bezogen werden. 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. Zum Beispiel stehen in der `pom.xml`, der Name des Projektes, die Version, die Art des Paketes, das später erstellt werden soll, die URL, die sich auf das Repository bezieht und die Abhängigkeiten, die das Projekt mit sich bringt und aufgelöst werden sollen, wenn das Projekt erstellt wird.

settings.xml

Zusätzlich wird eine Konfigurationsverwaltung erstellt, die `settings.xml`. In ihr werden die Zugangsdaten für den Repository-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 erstellt. Zusätzlich wird noch ein weiterer Ordner erstellt. Dieser heißt `.m2` 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 Benutzen von Maven nicht alles aus dem entfernten Repository heruntergeladen wird. Maven greift nur noch auf das lokale Repository zu, sofern die benötigten Pakete dort schon vorhanden sind. Wenn nicht, werden diese automatisch heruntergeladen.

IDE

Es gibt für Eclipse [12] und NetBeans [13] Plugins, um Maven in den IDEs zu integrieren und dort auch zu nutzen.

Man sieht also, dass mit wenigen Konfigurationseinstellungen die einzelnen Maven-Phasen (vgl. Abschnitt 3.2) verändert werden können. Aber diese Einstellungen ändern nichts am automatisierten Vorgehen von Maven. Dieser bleibt immer gleich.

Maven basiert auf einer Plugin-Architektur, die es erlaubt verschiedene Plugins für verschiedene Anwendungen auf das Projekt anzuwenden. Dadurch ist Maven sehr einfach erweiterbar.

Was Maven nicht ist

Zum Schluss noch einige Dinge darüber was Maven nicht ist. Maven ist keine Erweiterung des Build-Tools Ant. Maven stellt nach Sichten der Entwickler 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 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. Dabei startet der Lebenszyklus bei Maven erst in der Implementierungsphase. Phasen, die vor der Implementierung stattfinden, wie Spezifikation und Entwurf, werden von Maven nicht berücksichtigt. Diese werden im folgenden Abschnitt beschrieben.

3.2 Die Phasen Mavens

Kompilierungsphase In dieser Phase 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 Klassen werden dort beim Erstellen des Projektes hineingelegt.

Testphase In dieser Phase 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 aus dem Quellcode ein Paket erstellt. In der `pom.xml` kann man angeben, was für eine Art von Paket (jar, exe etc.) in dieser Phase erstellt werden soll. Die Datei wird in einem standardisiertem Verzeichnis abgelegt.

Integrations-Test In dieser Phase wird durch ein Plugin das fertige Paket in eine Umgebung integriert und getestet.

Verifizierungsphase In dieser Phase wird das Paket überprüft, ob es bestimmte Dateien/Ordner enthält und prüft den Inhalt auf ihre Struktur.

Installationsphase In dieser Phase wird das Paket in das lokale Maven-Repository 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.

3.3 Vergleich Maven, make und ant

Maven ist das zweite Buildprojekt von Apache. Das erste Buildprojekt ist *ant* [14]. Es existieren weitere Build-Tools, wovon *make* [15] das prominenteste ist. *Make* ist Teil des POSIX-Standards, dessen gegenwärtige Bezeichnung IEEE Std 1003.1, 2004 Edition lautet.

Apache-Ant

Apache Ant ist ein Java-basiertes Werkzeug 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 dienen zum Bauen des Programmes. Da es sich um eine XML-Datei handelt ist diese nicht von Tabulatorzeichen oder sonstigen Zeichen abhängig. Dies ist eine Verbesserung gegenüber dem alten Build-Tool make.

make

Make liest ein sogenanntes *Makefile*, in dem der Übersetzungsprozess von Programmen strukturiert erfasst ist, aus. 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 Maven, 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

Definition

Hudson [16] 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 [17] eine bedeutende Rolle. Denn durch die Agilität der Software wird die permanente Integration möglich. Das Extreme Programming [17] ist ein Beispiel für solch eine Agilität. 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 der Test fehl schlägt, 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 [16]. Hudson ist in Java geschrieben und läuft auf einem beliebigen Servlet-Container. Es werden sämtliche gängigen Build-Tools unterstüzt, wie Apache-Ant oder Apache-Maven. Außerdem kann Hudson den Quellcode automatisch aus einem SVN- oder CVS-Repository auschecken. Durch Plugins ist Hudson erweiterbar, diese bieten zum Beispiel eine Anbindung an „GoogleCode “. 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 den 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

In der Ausarbeitung habe ich drei gängige Werkzeuge beschrieben, die heutzutage als Standard gelten, wenn man größere Projekte entwickeln möchte.

Subversion ist nicht mehr als Werkzeug wegzudenken. Es bietet eine stabile Grundlage zur Sicherung und zur Verwaltung der Dateien. Dadurch kann eine große Softwaregruppe gleichzeitig an einem Projekt arbeiten. Und durch das ausgeprägte Loggen kann man stets nachvollziehen, welche Änderungen vorgenommen wurden. Zum anderen dient Subversion ebenfalls als Nachweis für die Arbeit eines Entwicklers. Durch weitere Programme kann man sich die kompletten svn-Daten auflisten lassen. Dabei sieht man welcher Entwickler welche Dateien hochgeladen hat. Ob der Inhalt der hochgeladenen Dateien sinnvoll ist, das kann Subversion nicht kontrollieren. Daher ist dieser Nachweis keine hundertprozentige Sicherheit dafür, ob der Entwickler effektiv gearbeitet hat. Man kann es aber zur Bewertung hinzuziehen.

Maven erleichtert einige Aufgaben eines Entwicklers oder einer Entwicklergruppe. Der standardisierte Software-Lebenszyklus wird automatisiert und unterschiedliche Strukturen werden aufgehoben. Dadurch entsteht ein einheitliches Programm, das 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, das permanent bei neuen Änderungen die Anwendung neu baut und testet, ist Maven ein weiteres Tool, mit dem Entwickler sehr viel Zeit sparen können. Diese Zeit können die Programmierer in Phasen investieren, 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 bereits benutzt. Ob sich Maven und Hudson integrieren lassen wird sich entscheiden. Mein persönlicher Wunsch wäre es, diese Tools zu benutzen, da sie uns sehr viel Arbeit abbnehmen würden. Außerdem würden sie uns zur frühen Fehlererkennung zwingen. Das erspart uns sehr viel Aufwand im Nachhinein bei der Fehlerbehebung und wir minimieren dadurch das Risiko des Scheiterns des Projekts

Literatur

1. Online-Lexikon, S.M.: (2009)
2. Ben Collins-Sussman, Brian W. Fitzpatrick, C.M.P.: Version Control with Subversion: For Subversion 1.5: (Compiled from r3305). TBA (2008)
3. Budszuhn, F.: CVS. (Galileo Computing)
4. (http://rapidsvn.tigris.org/)
5. (http://subcommander.tigris.org/)
6. (http://www.syntevo.com/smartsvn/index.html)
7. (http://tortoisesvn.tigris.org/)
8. (http://versionsapp.com/)
9. (http://subclipse.tigris.org)
10. Popp, G.: Konfigurationsmanagement mit Subversion, Ant und Maven. dpunkt.verlag (2006)

11. Vincent Massol, Jason van Zyl, B.P.J.C.C.S.: Better builds with Maven. Exist Global (2008)
12. (http://maven.apache.org/eclipse plugin.html)
13. (http://www.netbeans.org/kb/articles/subversion preview.html)
14. Matzke, B.: Ant. dpunkt.verlag (2003)
15. Andrew Oram, S.T.: Managing Projects with make. O'Reilly und Associates, Inc. (1986)
16. (http://hudson.gotdns.com/wiki/display/HUDSON/Meet+Hudson)
17. Jochen Ludewig, H.L.: Software Engineering. dpunkt.verlag (2006)

Alle URLs wurden zuletzt am 4. Februar 2009 geprüft.