1 Einleitung

Warum schreibt ein Softwarearchitekt ein Buch über Konfigurationsmanagement? Diese Frage – gerne noch mit dem Zusatz versehen, ob das Thema nicht etwas trocken wäre – musste ich schon im Vorfeld der ersten Auflage dieses Buches regelmäßig beantworten. Tatsächlich ist der Aufgabenkomplex, der sich hinter dem Schlagwort »Konfigurationsmanagement« (KM) verbirgt, ein eher ungeliebtes Kind der IT-Branche. Konfigurationsmanagement-Prozesse gelten als praxisfern, unproduktiv und teuer. Dies nicht ganz zu Unrecht, da derartige Prozesse oft unternehmensweit umgesetzt werden und entsprechend »schwergewichtig« ausgelegt sind. Die spezielle Situation eines einzelnen Projektes und die daraus resultierenden Probleme und Anforderungen haben in diesem Umfeld keine Priorität.

Tatsache ist allerdings auch, dass kein Projekt ohne die grundlegenden Verfahren des Konfigurationsmanagements auskommt. So ist die Verwendung eines Versionskontrollsystems wie z. B. Git oder eben Subversion heutzutage selbstverständlich. Ähnliches gilt für Werkzeuge zur Projektautomatisierung und zur Durchführung von Tests. Leider ist jedoch mit der Entscheidung für den Einsatz von einzelnen Werkzeugen in vielen Projekten das Ende der Fahnenstange erreicht. Weitergehende Richtlinien und Hilfsmittel existieren nicht, und deren Erstellung ist im Projektbudget auch nicht vorgesehen.

Trotzdem wurde in jedem Projekt, an dem ich als Architekt beteiligt war, eine leidenschaftliche und zeitintensive Debatte über Projektstruktur, Build-Prozess, Releases, Check-in-/Check-out-Richtlinien, Verwendung von Kommentaren und parallele Entwicklungszweige geführt. Und jedes Mal hat diese Diskussion zu Verzug in der Planung und zu erhöhten Kosten geführt.

Auch die vierte, aktualisierte Auflage des Buches hat den Anspruch, die Leserinnen und Leser beim »Ausfechten« der oben genannten Debatten im Projekt zu unterstützen. Ich beschreibe Konfigurationsmanagement als eine pragmatische Disziplin zur Erleichterung des

Projektalltags. Dieser Ansatz ist bisher eher selten anzutreffen. Stephen Berczuk stellt in [Berczuk02] sehr treffend fest, dass viele Konfigurationsmanagement-Prozesse die Projektteams eher behindern als unterstützen. Eine Ursache hierfür sei, dass KM-Prozesse auf den Anforderungen des Managements basieren und die täglichen Bedürfnisse der Teammitglieder schlicht ignoriert werden.

Ich habe alle in diesem Buch beschriebenen Aufgaben und Verfahren des Konfigurationsmanagements daher explizit aus dem Blickwinkel eines Softwarearchitekten und Entwicklers beschrieben. Dabei sind unweigerlich einige Aspekte der »reinen Lehre« des Konfigurationsmanagements auf der Strecke geblieben. Ich gehe aber davon aus, dass Sie als Leserin oder Leser dieses Buches mit dieser Tatsache leben können.

1.1 Wer dieses Buch lesen sollte

Zielgruppe dieses Buches sind technische Projektleiter, Softwarearchitekten und Entwickler, die für ihre Teams das Rad nicht zum x-ten Mal neu erfinden wollen. Viele von Ihnen werden sicherlich die von mir bereits erwähnten Diskussionen um Projektstruktur, Versionskontrollsysteme und die Verwaltung von Änderungsanforderungen aus eigener Erfahrung kennen. Die Anleitungen, Beispiele und Hinweise in den folgenden Kapiteln werden Ihnen helfen, diese Phase im Projekt schnell hinter sich zu bringen.

Was bringt Ihnen die Lektüre des Buches?

Neben dieser Hilfestellung hoffe ich, Ihnen zusätzliche Anregungen für einen effizienteren und angenehmeren Projektalltag geben zu können. Oft sind nur kleine Änderungen und Erweiterungen notwendig, um die Infrastruktur eines Projektes entscheidend zu verbessern. Wenn Sie mit Hilfe dieses Buches an der einen oder anderen Stelle im Projekt die Arbeit Ihres Teams erleichtern können, hat sich die Lektüre vermutlich schon gelohnt.

Vorausgesetzte Kenntnisse

Ich wende mich mit diesem Buch an Praktiker aus der Softwareentwicklung und setze daher eine Reihe von Kenntnissen voraus. Die Beispiele im Buch sind fast durchgängig so ausgelegt, dass sie von einer Shell-Umgebung aus nachvollzogen werden können. Ich werde nicht näher auf die Einrichtung einer solchen Umgebung eingehen und gehe davon aus, dass der Umgang mit Shells Ihnen keine Probleme bereitet. Die drei eingesetzten Werkzeuge Subversion, Maven und Redmine müssen Sie hingegen nicht kennen. Zwar ersetzt dieses Buch keine Benutzerdokumentation, doch die grundlegenden Konzepte und die zur Umsetzung eines KM-Prozesses notwendigen Funktionen werde ich im Verlauf des Buches beschreiben. Auf die Abgrenzung zur frei

verfügbaren Dokumentation für die Werkzeuge werde ich später noch eingehen.

Vorkenntnisse für die Einführung und die Subversion- bzw. Redmine-Kapitel

Für das Verständnis der theoretischen Einführung, des Subversion- und des Redmine-Kapitels setze ich keine Kenntnisse in bestimmten Programmiersprachen oder Entwicklungsumgebungen voraus. Zwar tauchen an einigen Stellen Java-Quellcodedateien auf, diese dienen jedoch lediglich als Platzhalter und können problemlos durch C#, Ruby oder Ihre persönliche Lieblingssprache ersetzt werden.

Vorkenntnisse für das Maven-Kapitel

Im Gegensatz zu Subversion und Redmine wird Maven hauptsächlich in Java-Projekten verwendet. Daher habe ich alle Beispiele im entsprechenden Kapitel auf diese Programmiersprache ausgelegt. Ich gehe in diesem Teil des Buches davon aus, dass Sie mit der Java-Entwicklung im Prinzip vertraut sind. Dies betrifft weniger die Kenntnis der Sprache selbst, denn es gibt im genannten Kapitel nur sehr wenige Beispiele, in denen wirklich Java-Quellcode zu sehen ist. Wichtiger ist, dass Sie mit den grundlegenden Techniken bei der Erstellung eines Java-Programmes vertraut sind, also beispielsweise mit dem Aufruf des Java-Compilers und der Generierung einer *jar*-Datei.

1.2 Warum Subversion, Maven und Redmine?

Ein falsches oder falsch eingesetztes Werkzeug löst keine Probleme, sondern schafft neue. Die Fokussierung dieses Buches auf genau drei Werkzeuge ist daher durchaus eine Erläuterung wert, schließlich passen Subversion, Maven und Redmine keinesfalls für alle denkbaren Softwareentwicklungsprojekte.

Ziel und Anspruch des Buches ist es, wie schon erwähnt, ein Praxishandbuch für Entwicklungsteams zu sein. Ohne konkrete Beispiele und direkt umsetzbare Anleitungen ist dieses Ziel nach meiner Auffassung nicht zu erreichen. Eine Vorauswahl geeigneter Werkzeuge war unter diesem Gesichtspunkt unumgänglich. Die Entscheidung für Subversion, Maven und Redmine fiel dann leicht, da alle drei frei verfügbar und in der Entwicklergemeinde recht populär sind. Zudem sind die drei Tools jedes für sich und in der Kombination sehr leistungsfähig und flexibel.

Kommerzielle Produkte versuchen demgegenüber oft, einen vom Hersteller vorgegebenen Prozess umzusetzen, und versprechen dem Anwender dafür erhebliche Produktivitäts- und Qualitätsvorteile. Für viele kleine und mittlere Projekte sind diese Vorgaben jedoch oft überdimensioniert und in der Praxis schwierig umzusetzen. Hinzu kommt, dass die »großen« Konfigurationsmanagement-Produkte enorm teuer sind.

Freie Software

Allen drei Tools gemeinsam ist ihr Status als *freie Software*, d. h., es fallen für private und kommerzielle Nutzung keinerlei Lizenzkosten an[1]. Obwohl freie Software seit Jahren im großen Stil professionell eingesetzt wird, hält sich das hartnäckige Gerücht, dass kommerzielle Software leistungsfähiger, stabiler und sicherer sei als die frei verfügbaren Alternativen. Dieses Vorurteil ist meiner Meinung nach eben ein solches, d. h. durch Fakten und Erfahrungen nicht zu belegen. Dies gilt für freie Software ganz allgemein und im Speziellen für Subversion, Maven und Redmine als Konfigurationsmanagement-Tools. Auch das oft genannte Argument des fehlenden professionellen Herstellersupports für freie Software ist erfahrungsgemäß nicht praxisrelevant. Bei wirklich dringenden Problemen findet man für kommerzielle und freie Werkzeuge am schnellsten und zuverlässigsten im Internet Unterstützung. Jeder, der schon einmal probiert hat, abends oder am Wochenende den Support eines Herstellers zu erreichen, wird mir hier sicherlich zustimmen.

Kommerzielle Konfigurationsmanagement-Werkzeuge

Natürlich gibt es Situationen, in denen kommerzielle Konfigurationsmanagement-Produkte besser geeignet sind als die in diesem Buch verwendeten freien Werkzeuge. Beispielsweise bieten die großen KM-Werkzeuge einen Grad an Integration, dem drei separate Open-Source-Werkzeuge nichts entgegenzusetzen haben. Voraussetzung für den Einsatz eines kommerziellen Werkzeuges ist allerdings, dass die Philosophie und der im Produkt verdrahtete KM-Prozess zum Projekt passen.

Es bleibt hinzuzufügen, dass ich bisher sehr selten Situationen erlebt habe, in denen ein kommerzielles Werkzeug den freien Tools wirklich überlegen war. Meist hat die hohe Flexibilität der Open-Source-Software die Vorteile einer voll integrierten Lösung mehr als aufgewogen.

Verwendete Versionen

Die Beispiele im Buch basieren auf den folgenden Versionen von Subversion, Maven und Redmine:

- Subversion 1.7.8
- Maven 3.0.4
- Redmine 2.2.1

1. Tatsächlich gehen die Lizenzbedingungen der drei Tools sogar noch darüber hinaus, da sie zusätzlich die Veränderung der Quelltexte freistellen. Für Subversion und Maven gelten die Apache License Version 2.0 (siehe *http://www.apache.org/licenses*). Redmine wird unter der GPL (siehe *http://www.gnu.org/licenses/old-licenses/gpl-2.0.html*) veröffentlicht.

1.3 Abgrenzung und Begriffserläuterungen

Konfigurationsmanagement ist ein weites Feld und keinesfalls ausschließlich auf das Gebiet der Softwareentwicklung begrenzt. Die ersten KM-Verfahren wurden schon in den 60er-Jahren im militärischen Bereich entwickelt. Auch heutzutage spielen KM-Prozesse bei der Entwicklung von Hardware noch eine große Rolle. Im Automobilbau werden z. B. erhebliche Anstrengungen unternommen, die hohe Variantenvielfalt durch Produktdatenmanagement-Prozesse im Griff zu behalten. Im Kern verwenden diese Verfahren klassische Konfigurationsmanagement-Techniken.

Software-Konfigurationsmanagement

Für uns als Softwarearchitekten und -entwickler sind diese Ausprägungen des Konfigurationsmanagements jedoch nicht interessant, auf sie wird im Buch daher auch nicht eingegangen. Wenn im Folgenden von Konfigurationsmanagement (oder kurz KM) gesprochen wird, meine ich immer implizit *Software-Konfigurationsmanagement*. In der englischen Literatur wird hierfür der Begriff *Software Configuration Management* oder kurz *SCM* verwendet.

KM und Versionskontrolle

Wie wir in Kapitel 2 sehen werden, ist die Verwaltung von Dateien oder, allgemeiner gesprochen, von Konfigurationselementen in einem Repository eine der wichtigsten Aufgaben des Konfigurationsmanagements. Vielleicht werden aus diesem Grund die Begriffe *Versionskontrolle* und *Konfigurationsmanagement* oft synonym verwendet. Dies ist jedoch nicht korrekt. Versionskontrolle (manchmal auch *Versionsverwaltung* genannt) ist »nur« ein Teilbereich des Konfigurationsmanagements.

Konfigurationsmanagement-Prozess

Weiterhin ist wichtig zu wissen, was in diesem Buch unter einem *Konfigurationsmanagement-Prozess* verstanden wird. Die unterschiedlichen Auffassungen von KM weisen eine große Bandbreite auf. Sehr formale Ansätze, die auf die Einführung eines unternehmensweiten KM-Prozesses abzielen, stehen pragmatischeren gegenüber. Umfassende KM-Prozesse setzen eigene Organisationseinheiten voraus und definieren diverse, KM-spezifische Rollen. Die Umsetzung zieht sich über einen langen Zeitraum, meist Jahre, hin und wird unter strategischen, unternehmenspolitischen Gesichtspunkten vorangetrieben. Wer Anregungen und Unterstützung für ein solches Vorhaben sucht, sollte einen Blick in die in Abschnitt 2.1.3 vorgestellten Standards werfen und z. B. in dem sehr guten Buch von Alexis Leon [Leon05] weiterlesen.

Leichtgewichtiger KM-Prozess

Für ein Praxishandbuch ist diese Herangehensweise nicht geeignet, auch wenn sie prinzipiell ihre Berechtigung hat. Daher wird im weiteren Verlauf des Buches ein leichtgewichtiger Ansatz präsentiert, der schnell und effizient in einem Softwareentwicklungsprojekt umgesetzt

werden kann. Der Konfigurationsmanagement-Prozess ist dabei immer auf ein konkretes Projekt bezogen. Er regelt, welche Werkzeuge wie von wem im Projekt eingesetzt werden. In einem unternehmensweiten Prozess wäre dieser Ansatz nur als untergeordneter Teilprozess geeignet. Die übergreifenden, strategischen und organisatorischen Gesichtspunkte werde ich nicht im Detail betrachten.

Abgrenzung zu der Dokumentation der Werkzeuge

Da der Fokus auf der Umsetzung eines pragmatischen KM-Prozesses mit Subversion, Maven und Redmine liegt, kann und soll dieses Buch kein Ersatz für die frei verfügbaren Benutzerhandbücher und spezialisierte Fachliteratur sein. Natürlich lernen Sie in den Praxiskapiteln den Umgang mit den drei Werkzeugen. Was Sie jedoch nicht finden werden, sind lange Listen mit Funktionsreferenzen. Ich werde stattdessen im Praxisteil mehrfach auf die frei verfügbare Online-Dokumentation von Subversion, Maven und Redmine verweisen.

1.4 Aufbau des Buches

Ich habe dieses Buch bewusst nicht als lose Sammlung einzelner Tipps und Tricks geschrieben. Konfigurationsmanagement ist ein Prozess und besteht aus miteinander verwobenen Teilbereichen. Dieser Ansatz findet sich auch im Aufbau des Buches wieder. Zur Umsetzung einer leistungsfähigen Projektautomatisierung muss beispielsweise die Projektstruktur von vornherein »richtig« aufgebaut werden. Dementsprechend machen wir uns über diesen Punkt Gedanken, bevor es an die Erstellung von Build-Skripten geht. Ich empfehle daher, die einzelnen Kapitel der Reihe nach zu lesen.

Theoretische Grundlagen

Das Buch besteht aus einem theoretischen (Kapitel 2) und aus einem praktischen Teil (Kapitel 3 bis 6). In Kapitel 2 werden die Grundlagen des Konfigurationsmanagements und die im Praxisteil verwendeten Begriffe ausführlich erläutert.

Praktische Umsetzung

Der praktische Teil besteht aus ingesamt vier Kapiteln. In Kapitel 3 stelle ich die verwendeten Werkzeuge Subversion, Maven und Redmine kurz vor. Die Kapitel 4, 5 und 6 beschreiben die praktische Umsetzung jeweils eines Teils des KM-Prozesses.

1.5 Beispielprojekt e2etrace

Ein Buch wie dieses lebt von Beispielen. Will man als Autor sicherstellen, dass die Beispiele der Leserschaft plausibel erscheinen, sollte man tunlichst alles, was man beschreibt, auch selbst ausprobieren. Da ich persönlich nur sehr ungern nutzlosen Code im Allgemeinen und Einkaufswagen-Beispiele im Besonderen schreibe, basieren alle Beispiele

im Buch auf dem Projekt *e2etrace*. Dieses Beispielprojekt, der Name steht übrigens für *End-to-End Tracing*, realisiert eine Java-Bibliothek mit Tracing-Funktionen. Im Gegensatz zu vielen anderen Bibliotheken, welche die Trace-Aufrufe lediglich in eine oder mehrere Logdateien schreiben, liegt der Schwerpunkt von *e2etrace* auf der Verfolgung einzelner Serviceaufrufe in einer verteilten Applikation. Die Idee ist, dass man nach der Ausführung eines Service den Trace als Teil des Ergebnisses erhält. Gerade wenn ein Service von mehreren, verteilten Komponenten implementiert wird, ist dies ein entscheidender Vorteil. Denn andernfalls müsste man sich die einzelnen Trace-Schritte aus den diversen Logdateien auf mehreren Plattformen mühsam zusammensuchen. Erfahrungsgemäß klappt dies schon bei geringer Last auf dem System nur noch mit erheblichem Aufwand.

Wenn Sie Interesse an *e2etrace* haben, finden Sie den Quellcode und die Dokumentation unter *http://www.e2etrace.org*. Für das Verständnis des Buches spielt die Funktionalität des Beispielprojektes keinerlei Rolle.

1.6 Konventionen

Deutsche und englische Fachwörter

Ich verwende im Buch hauptsächlich deutsche Fachwörter. Wenn keine allgemein akzeptierte deutsche Übersetzung existiert oder der Bezug zur Dokumentation der Werkzeuge durch die Übersetzung erschwert wird, habe ich es bei den englischen Begriffen belassen. Sollte meiner Ansicht nach eine passende deutsche Übersetzung existieren, gebe ich diese bei der erstmaligen Verwendung eines englischen Begriffes in Klammern an.

Typografie

Im Text verwende ich die *kursive Schriftlage* ganz allgemein als Hervorhebung. Ausschnitte bzw. Verweise auf Quelltext- und Shell-Beispiele werden im Fließtext durch die Schriftart `Letter Gothic` gekennzeichnet.

Quelltext-Beispiele

Gerade in den Praxiskapiteln finden Sie umfangreiche Quelltext-Beispiele in Form von durchnummerierten Listings:

Listing 1–1
Ein Quelltext-Beispiel

```
<!--Quelltext-Beispiel -->
<beispiel>
</beispiel>
```

Shell-Beispiele

Auch die Aufrufe der verwendeten Werkzeuge und deren Rückgaben beschreibe ich mit Hilfe von Beispielen. Shell-Beispiele folgen immer dem folgenden Muster:

```
> <Kommando>
Rückgabe
```

Der eigentliche Aufruf auf Shell-Ebene ist fett gedruckt, die Rückgabe hingegen normal. Wenn der Aufruf im Buch nicht in eine Zeile passt, breche ich ihn mit Hilfe des Zeichens \ um:

```
> <Kommando erste Zeile \
  zweite Zeile          \
  dritte Zeile>
Rückgabe
```

Ich habe alle Beispiele unter einer Windows-Shell entworfen. Im Text tauchen daher an einigen Stellen typische Windows-Pfade wie beispielsweise `D:\Projekte` auf. Da alle verwendeten Werkzeuge plattformunabhängig sind, sollten die Beispiele auch unter Linux funktionieren. Sie müssen in diesem Fall die Pfade entsprechend abändern.

Umgebungsvariablen

In einigen Abbildungen und auch an manchen Stellen im Fließtext verweise ich auf Umgebungsvariablen der Shell. Als Namensschema verwende ich in diesem Fall `%Variable%`.

1.7 Webseite zum Buch

Alle Beispiele aus dem Buch sowie die Literaturliste mit direkt »anklickbaren« Links finden Sie auf der Webseite zum Buch unter *http://www.km-buch.de*.

1.8 Danksagung

Ein Buch zu schreiben ist das genaue Gegenteil von meiner gewohnten Arbeitsweise, dem Teamwork. Als Autor sitzt man bisweilen sehr alleine vor dem digitalen Stück Papier. Daher ist jede Form der Unterstützung ungemein wertvoll. Mein Dank gilt allen Lesern der bisherigen Auflagen, die mir mit ihren Kommentaren wertvolle Hinweise für die vierte Auflage gegeben haben. Besonders danken möchte ich auch meinem Lektor René Schönfeldt vom dpunkt.verlag für die mittlerweile schon jahrelange gute Zusammenarbeit.

Gunther Popp
München, im April 2013