2 Automatisierte Integration mit Anthill & Co.

Thomas Herrmann

2.1 Überblick

Das vorliegende Kapitel basiert auf dem am 8. Oktober 2002 beim GI Arbeitskreis Java gehaltenen Vortrag „Automatisierte Integration - Toolunterstützter Build- und Integrationsprozess für Java-Projekte“.

Es beschreibt zunächst einige der Probleme, die bei der Arbeit an Softwareprojekten in kleinen und großen Teams auftreten, insbesondere, wenn es sich hierbei um verteilte Teams handelt.

Anschließend werden Konzepte und Lösungsansätze für einige dieser Probleme vorgestellt. Die präsentierte Entwicklungsumgebung basiert dabei ausschließlich auf frei verfügbaren Werkzeugen, die zum großen Teil im Rahmen von Open-Source-Projekten weiterentwickelt werden.

Der Einsatz derartiger Werkzeuge bietet sich gerade in kleineren Unternehmen und Projektteams an, da diese einerseits kostengünstig (da frei verfügbar) sind und andererseits (durch die Verfügbarkeit des Source-Codes) im Bedarfsfall einfach an das eigene Umfeld angepasst werden können.

Der Hauptteil des Kapitels widmet sich dann dem Softwarewerkzeug **Anthill**, das im Unternehmen des Autors seit einigen Monaten erfolgreich für die Integration der Arbeitsergebnisse eingesetzt wird.

2.2 Probleme bei der Softwareentwicklung im (virtuellen) Team

Jedes größere Softwaresystem wird heutzutage in mehr oder weniger großen Teams entwickelt. Nach der Festlegung der Gesamtarchitektur des Systems durch einen Software-Architekten oder ein kleines Architektur-Team erfolgt die Realisierung in der Regel unabhängig voneinander durch mehrere Softwareentwickler oder Entwicklerteams.

Trotz der (meist zu Projektbeginn) definierten Vorgaben für die Entwicklung gibt es diverse Probleme bei der Zusammenarbeit eines Teams von Entwicklern.

So gibt es immer **persönliche Vorlieben der Entwickler**, z.B. in Bezug auf den eingesetzen **Editor** oder die eingesetze **IDE** oder auch das **Code-Layout** (Einrückungen, Klammersetzung etc.).

Das Zusammenspiel der Softwarekomponenten bedarf der genauen **Abstimmung von Schnittstellen** zwischen diesen Komponenten. Theoretisch kann dies jeweils vor der Entwicklung einzelner Komponenten erfolgen. Die Praxis zeigt jedoch, dass oftmals während der Entwicklung weitere Funktionalitäten benötigt werden, die Schnittstellenänderungen und/oder -erweiterungen hervorrufen.

In großen Projekten erfolgt diese Abstimmung im Idealfall regelmäßig in entsprechenden Abstimmungsgremien. Die dadurch entstehenden Zeitverzögerungen und der zusätzliche Aufwand führen jedoch nicht selten zu „Übergangslösungen“, die saubere Software-Architekturen unterlaufen und Funktionalitäten falsch zuordnen, nur um die festzementierten Schnittstellen nicht ändern zu müssen.

Ein weiteres Problem stellt die **Vielzahl von Versionen** einzelner Source-Dateien und auch ganzer Module dar. Hier verliert man ohne den Einsatz geeigneter Verwaltungstools schnell den Überblick. Insbesondere kann die Kompatibilität zwischen Objekten oft nicht einfach überprüft werden.

Bei der Arbeit an einem Modul kann der Entwickler relativ einfach lokale Fehler ausschließen, z.B. durch Nutzung von Unit-Tests. Das Auffinden **nicht-lokaler Fehler (Integrationsprobleme)** ist jedoch schon wesentlich schwieriger, da der einzelne Entwickler oftmals gar nicht die Infrastruktur zum Betrieb und damit zum Test des Gesamtsystems zur Verfügung hat (man denke z.B. an verteilte Anwendungen). Eine möglichst frühzeitige Integration der Entwicklungsergebnisse ist daher wünschenswert.

2.3 Lösungsideen und Lösungskonzepte

Um die angesprochenen Probleme zu vermeiden oder zumindest abzuschwächen, haben wir einige Strategien angewandt.

Zunächst wollten wir den Entwicklern die notwendige Freiheit in der Auswahl „ihrer“ Werkzeuge geben. Entwickler sollen das verwenden, was sie wollen.

Nachdem wir uns in der Entwicklung ausschließlich auf Java-Anwendungen konzentrieren, ist unsere einzige Anforderung die Verwendung von Standard-Java-Werkzeugen. Dies bedeutet z.B., dass die Erstellung des Systems keine IDE-spezifischen Bibliotheken erfordern darf.

Eine Vereinheitlichung des Source-Codes erfolgt einerseits durch Programmierrichtlinien. Anstatt hier eine eigene Definition vorzunehmen, stützen wir uns jedoch auf die von Sun veröffentlichten Code-Konventionen (siehe `http://java.sun.com/docs/codeconv/`).

Die Einhaltung dieser Konventionen kann durch den Einsatz des Werkzeuges **Checkstyle** (siehe unten) überprüft werden. Darüber hinaus verwenden wir die Möglichkeit, Source-Code mit Hilfe von Tools gemäß den Programmierrichtlinien zu formatieren.

Tabelle 2.1: *Im Entwicklungsprozess eingesetzte Werkzeuge*

Aufgabe	**Werkzeug**
Versionsmanagement	CVS
Buildprozess	Ant
Integration	Anthill
Unit-Tests	JUnit
Browsebare Sourcen	Java2HTML
Code-Check	Checkstyle
Doku-Update	PHPWiki

Anstatt einmal festgeschriebene Schnittstellen in Stein zu meißeln, wird in der Entwicklung die Evolution von Schnittstellen propagiert. Um die dadurch hervorgerufene häufige Änderung der Schnittstellen in den Griff zu bekommen, ist eine laufende Integration der Module notwendig. Dieses Vorgehen wurde unter dem Namen **Continuous Integration** bekannt.

Unsere grundlegende Idee bei der Softwareentwicklung ist, **Fehler und Probleme möglichst frühzeitig und vollständig zu erkennen**.

Hierzu haben wir versucht,

- einheitliche Strukturen und Prozesse zu definieren,
- kleine, überschaubare Einheiten zu bilden und
- eine laufende Integration durchzuführen.

Um dieses Ziel zu erreichen, ist die Verwendung mehrerer Werkzeuge notwendig, die im nächsten Abschnitt dargestellt werden. Eine zentrale Rolle spielt dabei das Integrationswerkzeug **Anthill**.

2.4 Werkzeuge

In der Tabelle 2.1 sind die von uns für die verschiedenen Themenbereiche eingesetzten Werkzeuge dargestellt. Im Folgenden wird ein kurzer Überblick über diese Werkzeuge gegeben. Eine genauere Betrachtung würde den Umfang dieses Kapitels sprengen. Weitere Informationen sind unter den am Ende dieses Kapitels aufgeführten Links zu finden.

Eine ganz zentrale Rolle spielt hierbei das Source-Repository. Hierfür kommt bei uns **CVS** zum Einsatz. Dieses seit vielen Jahren bewährte Versionsmanagement ist geeignet für kleine bis mittlere Teams und zeigt seine Stärken insbesondere bei verteilten Teams. Es besitzt natürlich nicht die Mächtigkeit der „großen“ Versions- und Change-Managementsysteme wie ClearCase oder ChangeSynergy, leistet aber bei überschaubaren Projekten sicher und effizient seine Dienste. CVS, selbst ein Open-Source-System, wird daher in beinahe allen bekannten Open-Source-Projekten verwendet.

Zur Übersetzung der Java-Sourcen kommt **Ant** zum Einsatz, das sich in diesem Bereich als Standard-Werkzeug etabliert hat. Ant ist auf allen Plattformen, die eine Java-VM unterstützen, lauffähig und bietet damit die besten Voraussetzungen als plattformunabhängiges Build-Werkzeug. Insbesondere läßt sich Ant auch in einige populäre integrierte Entwicklungsumgebungen (IDEs) integrieren.

Die Integration der Module wird durch **Anthill** unterstützt. Anthill wird weiter unten in diesem Kapitel im Detail vorgestellt.

Um Module nur dann freizugeben, wenn sie nicht nur syntaktisch korrekt sind, sondern auch ein Mindestmaß an definierter Funktionalität korrekt implementieren, werden Modultests mit Hilfe von **JUnit** durchgeführt. Die Ausführung der Unit-Tests wird ein Bestandteil des Standard-Build-Prozesses werden.

Der Einsatz von **Java2HTML** ermöglicht die Veröffentlichung der Sourcen als Web-Seiten. Damit können Entwickler Source-Code von Modulen einsehen, die sie gar nicht selbst als Source-Module verfügbar haben.

Zur Prüfung des Source-Codes, insbesondere in Bezug auf Einhaltung der Programmierrichtlinien, setzen wir **Checkstyle** ein. Über Parametereinstellungen kann Checkstyle sehr gut an die zu überprüfenden Richtlinien angepasst werden.

Um das Problem fehlender oder veralteter Dokumentation zu mildern, kommt **PHPWiki** zum Einsatz. PHPWiki ist eine der vielen verfügbaren Implementierungen eines WikiWikiWeb. Die Grundidee hierbei ist die Möglichkeit, Webseiten jederzeit (durch einen definierten Nutzerkreis) ändern zu lassen. Hiermit sind Notizen und die Dokumentation von Änderungen einfach und ohne großen Zeitaufwand möglich.

2.5 Einheitliche Modulstruktur

Zur Vereinfachung und Automatisierung des Build- und Integrationsprozesses ist die Vereinheitlichung der einzelnen Module von großem Vorteil.

Daher entschlossen wir uns schon frühzeitig, eine möglichst einheitliche Modulstruktur und insbesondere einen einheitlichen Buildprozess zu definieren.

Neben der Möglichkeit, Dinge leichter Automatisieren zu können, bietet die einheitliche Modulstruktur auch ein schnelles Zurechtfinden, auch in fremden Modulen.

2.5.1 Modultypen

Zunächst wurden von uns die wichtigsten Modultypen identifiziert.

Diese sind:

- **Bibliotheksmodule**, die verwendete 3rd-Party-Software kapseln. Diese Module (z.B. Log4J) werden in der Regel nur in compilierter Form als jar-Dateien und nicht im Source-Code verwaltet. Um die jeweils eingesetzte Version verwalten zu können, pflegen wir sie dennoch im CVS. Die Struktur der Bibliotheksmodule ist im nächsten Abschnitt dargestellt.

- **(Java-) Entwicklungsmodule**, die als Ergebnis jar-Files und gegebenenfalls weitere Dateien wie z.B. Property-Dateien, Grafiken etc. erstellen. Die Übersetzung erfolgt durch ein standardisiertes Ant-Build-Script. Die Struktur der Java-Entwicklungsmodule ist ebenfalls im nächsten Abschnitt beschrieben.
- **Web-Module**, die neben Java-Code auch JSP-Dateien und Servlets enthalten. Diese Module werden von „normalen“ Java-Entwicklungsmodulen unterschieden, da das Build-Script anders ist.

Neben den genannten Modultypen wollen wir zukünftig eine weitere Klassifizierung der Module durchführen und die jeweils am besten geeignete Modulstruktur definieren.

2.5.2 Beispiele Modulstruktur

Nachfolgend wird die Struktur der Bibliotheksmodule und der Entwicklungsmodule beispielhaft dargestellt. Durch die Vereinheitlichung der Modulstruktur und der darauf aufbauenden Skripten konnten wir den Aufwand zur Einführung neuer Module auf ein Minimalmaß reduzieren.

Der Aufwand zur Einführung eines Moduls (inklusive der Aufnahme in Anthill) als eigenständiges CVS-Modul liegt inzwischen bei weinigen Minuten.

Bibliotheksmodule.

```
<dev>
  |
  +-- <modul>       Verzeichnis des CVS-Moduls
  |     |
  |     +--- export Exportierte Dateien des 3rd-Party- Moduls
  |     |                  (im CVS)
  |     +--- src    buildNumber
  |               build.xml
  |               release.xml
  |
  +-- <modul>.export
```

Entwicklungsmodule.

```
<dev>
  |
  +-- <modul>    Verzeichnis des CVS-Moduls
  |     |     buildNumber
  |     |     import.list
  |     |     module.properties
  |     |
  |     +--- export  Exportierte Dateien des Moduls
  |     |                  (durch build erstellt)
```

```
|     +--- import  Importierte Dateien des Moduls
|     |                  (durch build gefüllt)
|     +--- doc     Modul-Dokumentation
|     |               (im CVS und generiert)
|     +--- class   Übersetzte Klassen
|     |                (durch build gefüllt)
|     +--- src     Basis-Sourceverzeichnis
|                     build.xml
|                     release.xml
|                     Version.java.template
|
+-- <modul>.export
```

2.6 Das Integrationstool Anthill

Die Idee, die Integration der Softwareentwicklungsergebnisse automatisiert durchzuführen, wurde von uns schon lange verfolgt. Durch den Start eines größeren Projektes Anfang 2002, bei dem wir u.a. mehrere freie MitarbeiterInnen beschäftigten, wurde der Leidensdruck hierfür natürlich nochmals erhöht.

Nach einigen Recherchen wurde beschlossen, das Werkzeug **Anthill** hierfür in einer Probeinstallation einzusetzen. Einige Wochen später wunderten sich alle Beteiligten, wie man vorher eigentlich ohne ein derartiges Werkzeug klargekommen war.

2.6.1 Das Konzept von Anthill

Die Grundidee von Anthill ist **Continuous Integration**, das heißt die laufende Integration der entstehenden Projektergebnisse. Durch diese laufende Integration sollen Fehler im Zusammenspiel der einzelnen Module möglichst führzeitig erkannt werden.

Die **Reproduzierbarkeit** von Ergebnissen wird dabei durch Build-Protokolle, Protokollierung der Testergebnisse, Dokumentation und Metriken sowie Kennzeichnung der jeweils verwendeten Source-Dateien erreicht.

Nach erfolgreichen Builds erfolgt die Markierung, „Tagging“, aller in diesem Build verwendeten Sourcen mit einer Buildnummer. Diese wird bei jedem Build inkrementiert.

Um bei auftretenden Problemen jederzeit die verwendeten Programm-Sourcen reproduzieren zu können, haben wir eine Abfragemöglichkeit der Build-Nummer zur Laufzeit implementiert.

2.6.2 Arbeitsablauf mit Anthill

Der Ablauf, der durch Anthill implementiert wird, ist in 2.1 dargestellt.

Die Arbeitsschritte sind dabei die Folgenden:

- **Regelmäßige Prüfung auf Änderungen** durch Prüfung des Source-Code-Repository. Falls dort keine Änderungen erkannt wurden, wird die Verarbeitung beendet. Die Überprüfung erfolgt regelmäßig über sogenannte **Schedules**.
- **Update der Work-Area** mit der neuesten Version aus dem Source-Code-Repository und Update der geänderten Dateien.
- **Inkrement der Build-Nummer**, um diesen Build jederzeit eindeutig identifizieren zu können.
- **Tagging der Sourcen** mit der verwendeten Build-Nummer. Durch diese Markierung können jederzeit für jeden Build die hierfür verwendeten Sourcen reproduziert werden.
- **Build**, also Übersetzung des aktuellen Entwicklungsstandes. Idealerweise werden im Rahmen des Builds auch die Modultests durchgeführt.
- **Release** des Moduls, falls der Build und der Modultest erfolgreich verlaufen ist. In der Regel werden die Build-Ergebnisse dabei an eine zentrale Stelle (z.B. Intranet) kopiert, von wo sie von allen Entwicklern einsehbar sind.
- **Information per E-Mail** über das Build-Ergebnis an registierte Benutzer. Je nach Build-Ergebnis können die so informierten Entwickler gegebenenfalls Korrekturen an dem Modul vornehmen.

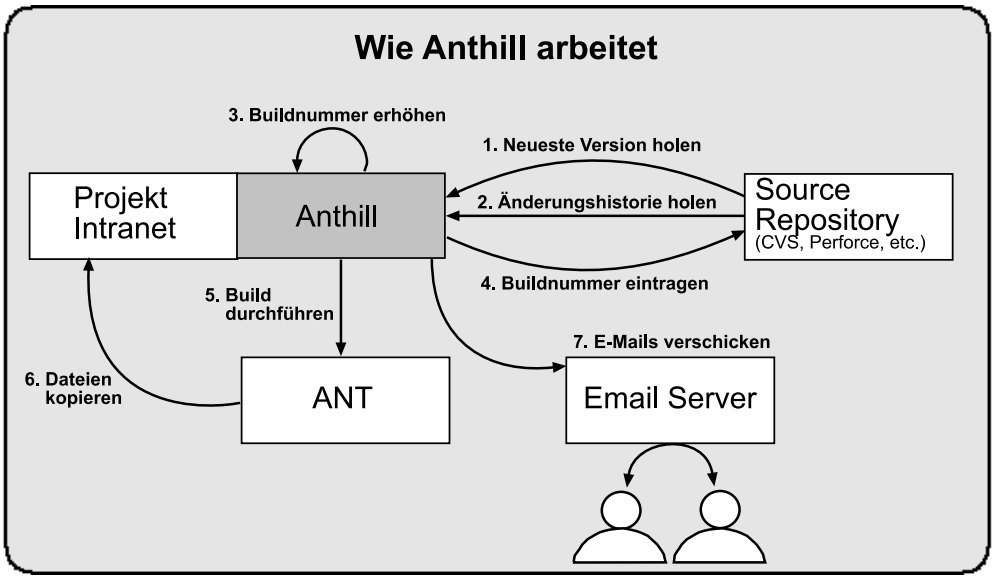

Abb. 2.1: *Arbeitsablauf mit Anthill*

2.6.3 Unterstützung durch Urbancode

Der Support durch Urbancode, dem Hersteller von Anthill, erfolgt über eine Mailingliste, zu der man sich auf der Urbancode-Website anmelden kann. Dieser Support ist in der Regel ganz ausgezeichnet. Die Probleme der Anwender werden sehr ernst genommen und Korrekturen von Fehlern sind oftmals im nächsten veröffentlichten Build enthalten.

Neben Fehlerbehebungen erfolgt aber auch die Umsetzung von Erweiterungswünschen sehr schnell, wenn es sich um Erweiterungen von allgemeinem Interesse handelt, die sich nahtlos in das Anthill-Konzept integrieren lassen.

2.6.4 Die Zukunft von Anthill

Im Dezember 2002 wurde von Urbancode angekündigt, dass das nächste Release (das bisher immer als Anthill 2 bezeichnet worden war) unter dem neuen Namen **Anthill Pro** als kommerzielles Produkt verfügbar sein würde.

Bei Anthill Pro handelt es sich um ein vollständiges Redesign von Anthill. Besonderer Wert wurde hierbei auf größtmögliche Flexibilität gelegt, sowohl in Bezug auf die eingesetzten Werkzeuge als auch im Hinblick auf Konfigurierbarkeit.

So kann der Build durch das Konzept sogenannter *builder* in Anthill Pro nicht nur mit Hilfe von *ant*, sondern auch mittels *make* erfolgen. Außerdem werden neben CVS auch ClearCase, Perforce, PVCS und Visual Source Safe über *RepositoryAdapter* unterstützt. Eine Integration mit Tools wie *Bugzilla* ist geplant.

Der Preis für diese kommerzielle Anthill-Version beträgt US-$ 1.299,-. Ein Preview-Release von **Anthill Pro** finden Sie auf der Website von Urbancode.

Dies bedeutet jedoch keinesfalls den Tod von **Anthill**, ganz im Gegenteil. Eine Nachfrage bei Urbancode ergab, dass es zukünftig als „richtiges“ Open-Source-Projekt weiterentwickelt werden soll. Bisher konnte zwar der Source-Code bezogen werden, Änderungen und Erweiterungen konnten aber nur über Urbancode eingebracht werden.

Zukünftig wird der Source-Code der Version **Anthill OS** (Open Source) in einem öffentlichen Source-Repository zur Verfügung stehen und kann somit von den hoffentlich vielen Entwicklern, die diese Version warten und erweitern werden, selbst verwaltet werden.

Es bleibt zu hoffen, dass die Anwendergemeinde von Anthill OS dieses hervorragende Werkzeug in Zukunft genau so aktiv weiterentwickeln wird, wie es bisher durch Urbancode selbst erfolgt ist.

2.7 Zusammenfassung

In den vorangegangenen Abschnitten wurde dargestellt, wie man gerade mit einfachen Mitteln und unter Einsatz frei verfügbarer Werkzeuge den Softwareentwicklungsprozess entscheidend verbessern kann.

Gerade das Konzept der kontinuierlichen Integration ermöglicht die frühzeitige Erkennung von Programmfehlern durch die schnelle Zusammenführung der Arbeitsergebnisse aller beteiligten Teammitglieder.

Die auf diese Weise frühzeitig identifizierten Probleme können dadurch zeitnah gelöst werden. Insgesamt ergibt sich damit ein schnellerer Entwicklungsprozess und gleichzeitig eine bessere Qualität der Entwicklungsergebnisse.

2.8 Links

Alle in diesem Kapitel angesprochenen Werkzeuge können kostenlos über das Internet bezogen werden (Stand der folgenden WWW-Links: 31.1.2003).

Anthill, das in diesem Kapitel vorgestellte Werkzeug, ist auf der Website von Urbancode unter `http://www.urbancode.com/projects/anthill/` zu finden.

Die aktuelle und geplante Funktionalität der kommerziellen Version **Anthill Pro** kann unter `http://www.urbancode.com/products/anthillpro/profeatures.jsp` nachgelesen werden.

Ant ist das für Java-Projekte am häufigsten verwendete Build-Tool. Es ist Teil des Apache-Projektes und unter `http://ant.apache.org/` zu finden.

CVS, Abkürzung für **C**oncurrent **V**ersion **S**ystem ist ein einfaches Versionsmanagementsystem, das in nahezu allen Open-Source-Projekten verwendet wird. Zu finden unter (`http://www.gnu.org/software/cvs/`).

Der „Standard“ für Unit-Tests in Java ist sicherlich **jUnit**, `http://www.junit.org/`.

Um aus Java-Programmcode navigierbare HTML-Seiten zu machen, bietet sich das Werkzeug **Java2HTML** unter `http://www.java2html.com/` an.

Checkstyle dient der Überprüfung von Java-Programmen gegen formale Kriterien, wie sie z.B. in Programmierrichtlinien definiert sind (`http://checkstyle.sourceforge.net/`).

Gerade in den Phasen der Ideenfindung kann die Kommunikation innerhalb eines Teams durch den Einsatz eines sogenannten „Wikiwebs“ unterstützt werden, wie es z.B. **PHPWiki** darstellt. Zu finden unter `http://phpwiki.sourceforge.net/`.

2.9 Über den Autor

Thomas Herrmann war schon während seines Studiums an der Technischen Universität München freiberuflich als Anwendungsentwickler tätig. Nach dem Abschluss als Diplominformatiker widmete er sich freiberuflich weiteren Aufgaben in der kommerziellen Anwendungsentwicklung. Die Schwerpunkte seiner Tätigkeit lagen in der Erstellung von Softwaremodellen und Softwarearchitekturen.

Zusammen mit seiner Frau Michaela gründete Thomas Herrmann 1998 die Teleteach GmbH (`http://www.teleteach.de/`), die seither als Beratungs- und Softwareunternehmen im damals gerade beginnenden eLearning-Markt tätig ist.

Die in diesem Kapitel beschriebenen Vorgehensweisen werden von Teleteach bei der Entwicklung der Teleteach Learning Suite (TLS) sowie zur Unterstützung von Kundenprojekten eingesetzt.