Professionelle Software-Entwicklungs-Umgebungen in der Hochschulausbildung

Boris Wickner, Bernd Müller

Fakultät Informatik
Ostfalia – Hochschule für angewandte Wissenschaften
Salzdahlumer Str. 46/48
38302 Wolfenbüttel
{bo.wickner,bernd.mueller}@ostfalia.de

Abstract: Mit der fortschreitenden Verbreitung agiler Software-Entwicklungsmethoden finden auch entsprechende Software-Werkzeuge Verbreitung in der professionellen Software-Entwicklung. Methoden und Werkzeuge der kontinuierlichen Integration unterstützen Entwicklungsprozesse und steigern ihre Effizienz. Subversion zur Versionsverwaltung, Ant oder Maven zur Build-Automatisierung, Jenkins zur kontinuierlichen Integration sowie Sonar als Metrik-Dashboard sind Werkzeuge, wie sie in vielen kommerziellen Entwicklungsprozessen eingesetzt werden. Weil alle genannten Werkzeuge als Open-Source-Systeme entwickelt werden, ist ihr Einsatz in der Hochschulausbildung sowohl unter einem finanziellen Aspekt als auch unter der Maßgabe einer einfachen Verwendung am heimischen studentischen PC von Interesse.

Wir beschreiben die Verwendung der Werkzeuge in einer Master-Veranstaltung der Fakultät Informatik der Ostfalia vor allem im Hinblick auf ihre technische Funktionalität. Studenten kommen früh mit den Werkzeugen der professionellen Software-Entwicklung in Kontakt. Die Qualität der Ausbildung erfährt einen deutlichen Mehrwert, da die Praxisnähe deutlich gesteigert wird.

1 Einleitung

Moderne Prozessmodelle in der Software-Entwicklung verbinden inkrementelle und iterative Modelle mit den Prinzipien der kontinuierlichen Integration. Die professionelle Projektarbeit verwendet zunehmend diese Methoden, um nach möglichst kurzer Zeit grundlegende Anforderungen an die Software in einer lauffähigen Version umgesetzt zu haben.

Dieses Vorgehen ermöglicht das frühe Erkennen von Fehlern in Anforderungen und deren Umsetzung und reduziert durch frühes Feedback an das Entwickler-Team Risiken und Kosten des Projekts.

Im Gegensatz zur klassischen Entwicklung nach dem Wasserfall-Modell – nach dem das Projekt strikt in Phasen aufgeteilt ist und die Fertigstellung erst in der Integrationsphase falsch umgesetzte Spezifikationen aufzeigt – setzten agile Prozesse verstärkt auf starke Kommunikation zwischen Projektleitung, Entwicklern und den Kunden.

Durch den Einsatz von Werkzeugen zur Unterstützung und Automatisierung von Teilprozessen wird die Effizienz des Vorgehens weiter erhöht.

2 Einzelsysteme

Die hier beschriebene Konfiguration setzt im Kern auf vier zentrale Komponenten, um agile Entwicklungsprozesse mithilfe des Konzepts der kontinuierlichen Integration zu unterstützen. Abbildung Abbildung 1 stellt dies schematisch dar.

Die Installation und Konfiguration der einzelnen Systeme kann mit wenig Aufwand durchgeführt werden, da für jede Komponente Plugins zur Integration in bestehende Infrastrukturen zur Verfügung stehen, wie beispielsweise die Benutzerautorisierung mittels eines zentralen LDAP-Servers.

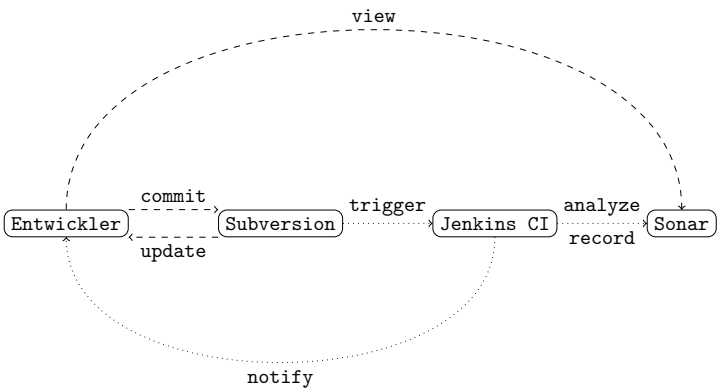

Abbildung 1: Schematische Darstellung der Infrastruktur

2.1 Versionsverwaltung

In der hier vorgestellten Infrastruktur ist die Versionsverwaltung die zentrale Schnittstelle für den einzelnen Entwickler. Nach einmaliger Konfiguration der weiteren Systeme löst ein Commit in die Versionverwaltung die Ausführung der weiteren Aktionen in der Infrastruktur aus.

Die auftragsbezogene Entwicklung stellt in vielen Bereichen die Anforderung, dass Entwicklungsverläufe über den gesamten Zeitraum nachvollziehbar sein müssen. Im Unternehmensumfeld hat sich *Subversion* als de facto Standard zur Versionsverwaltung etabliert. Durch die zentrale Führung des Projektstandes wird eine konfliktfreie Entwicklung

im Team ermöglicht und eine Schnittstelle für die weiteren unterstützenden Tools der Infrastruktur geschaffen.

Während der alltäglichen Entwicklungsarbeit besteht der Kontakt mit Subversion jedoch nicht nur im Updaten und Einpflegen der eigenen Arbeitskopie und Änderungen. Gerade in größeren Projektteams können Konflikte bei Änderungen des Quellcodes auftreten, die es zu beheben gilt. Ebenso ermöglicht der Versionsverlauf das komfortable Rückkehren zu einem früheren Stand, um die Ursache neu entstandener Fehler ausfindig machen zu können.

Aus dem Einsatz von Subversion haben sich *Best-Practices* und *Workflow-Modelle* entwickelt, mit denen sich zukünftige Entwickler auseinandersetzten und auskennen müssen. Nicht nur der sichere Umgang mit Konfliktsituationen bei Commits oder Updates, sondern die konsequente Umsetzung und Anwendung von *Tagging-* und *Branching-Modellen*, sowie der eigentliche Inhalt der Repositories sollte aus dem Stegreif beherrscht werden. Die Erfahrung des praktischen Einsatzes während der Ausbildung bereitet Studenten somit besser auf die spätere Tätigkeit vor.

Gerade der erste Kontakt mit den Prinzipien der Versionsverwaltung kann Unerfahrene schnell den Überblick verlieren lassen. Dezentrale Versionsverwaltungen, wie beispielsweise *GIT* oder *Mercurial*, bieten dem Team zwar sehr viel mehr Freiraum in der Gestaltung ihrer Zusammenarbeit. Der gewonnene Freiraum erhöht jedoch ebenfalls die Komplexität der Bedienung und das Auflösen von Fehlern in der Bedienung enorm. Darüber hinaus ermöglichen dezentrale Versionsverwaltungen auch das gezielte Manipulieren der History.

Aus diesem Grund werden sie in vielen Unternehmen noch nicht eingesetzt und von der Nutzung im Lehrbetrieb sollte abgesehen werden.

2.2 Build-Automatisierung

Während der Ausbildung im Bereich der Programmierung werden nicht nur grundlegende Kenntnisse im Umgang mit der Programmiersprache vermittelt, sondern auch der Umgang mit verbreiteten Entwicklungsumgebungen erlernt. Da jede integrierte Entwicklungsumgebung Möglichkeiten zum direkten Kompilieren, Testen und Ausführen der Software bietet, geht hierbei die Portabilität des Projektes verloren. Die Projekte sind gemäß der Entwicklungsumgebung strukturiert und werden nach deren Konfiguration verarbeitet. In der Entwicklung mit Eclipse werden beispielsweise Konfigurationen von Bibliotheken und Umgebungsvariablen – getrennt von der Projekt-Konfiguration – in der Workspace-Konfiguration gespeichert.

Um die Bindung an eine Entwicklungsumgebung zu lösen, bietet sich der Einsatz von Werkzeugen zur Automatisierung von Build-Prozessen an. Ähnlich dem weithin bekannten Tripel *./configure; make; make install* unter unixoiden Betriebssystemen, haben sich im Java-Umfeld die Werkzeuge Apache *Ant* und *Maven* etabliert.

Ant orientiert sich hierbei am Konzept von *GNU make*, das einzelne Tasks und Abhängig-

keiten untereinander beschreiben. Zu jedem Task werden die auszuführenden Programmaufrufe spezifiziert. Der Vorteil bei Ant liegt bei extrem fein konfigurierbaren Build-Prozessen. In den Prozess kann gezielt eingegriffen und an jeder Stelle praktisch beliebig erweitert werden. Der deutlichste Nachteil an dieser Methode ist der große Aufwand, um komplexe Prozesse abzubilden. Werden viele Phasen benötigt, wächst die Konfigurationsdatei schnell auf unüberschaubare Größen an.

Maven hingegen orientiert sich im Kern am Konzept der *Convention over Configuration*. Die Konvention in Maven sieht einen festen Build-Lifecycle vor, der aus Phasen zusammengesetzt wird. In diesen Phasen werden beispielsweise Quellen und Testfälle kompiliert, Testfälle ausgeführt oder das Projekt in einem JAR-Archiv verpackt. Die minimale Projektkonfiguration setzt lediglich die Angabe der zur Identifizierung eines Artefakts nötigen Attribute – *groupId*, *artifactId*, *version* und *packagingType* – voraus. Aus dem gewählten Packaging-Type ergibt sich die aus Konventionen bestehende Konfiguration des Build-Prozesses. Während dieses Schrittes wird die Ausführung von Maven-Plugins an Phasen im Build-Lifecycle gebunden. Da diese Konfiguration auf einer Konvention beruht, kann sie durch explizite Konfiguration in der Projektbeschreibung verändert und erweitert werden.

Dringt man weiter in die Möglichkeiten von Maven vor, so ergeben sich Möglichkeiten des Resource-Filterings. Mit diesen Techniken lassen sich infrastrukturbedingte Konfigurationen, beispielsweise Verbindungsparameter für Datenbanken, aus dem eigentlichen Projekt und dessen Konfiguration extrahieren und über Parameter zur Ausführungszeit des Builds injizieren.

Als recht neues Werkzeug findet *Gradle* derzeit viel Beachtung. Hier wurde versucht, die Vorteile von Ant und Maven zu vereinen, ohne jedoch die Nachteile zu übertragen. Gradle bringt alle Stärken von Maven – maßgeblich Convention over Configuration und die automatische Abhängigkeitsverwaltung – mit sich, setzt bei der Projektbeschreibung jedoch auf eine Domain Specific Language. Diese DSL ist stark an die Ruby-Syntax angelehnt und ermöglicht eine exakte Erweiterung des Build-Prozesses – direkt in der Projektkonfiguration.

Alle hier genannten Werkzeuge haben Vor- als auch Nachteile. Im späteren Berufsleben werden Absolventen ohne Zweifel nicht ohne den Gebrauch dieser Werkzeuge auskommen. Eine frühe Auseinandersetzung mit den Möglichkeiten der Build-Automatisierung bereitet maßgeblich auf die Projektarbeit im Berufsleben vor.

Ant hat in der Open-Source-Welt aber auch in der unternehmensbezogenen Software-Entwicklung bereits einen hohen Verbreitungsgrad erreicht. Maven gewinnt mittlerweile jedoch in vielen Projekten die Oberhand, da es sich als unkomplizierter Ansatz erweist, der auf festen Projektstrukturen und anderen Konventionen beruht. Da viele neue Frameworks im Testing-Bereicht konsequent auf die Nutzung in Maven-Projekten konzipiert werden, sollte bevorzugt Maven zum Einsatz kommen.[1]

[1] [PSCM08] bietet eine gute, kostenlose Einführung in den Umgang mit Maven.

2.3 Continuous-Integration-Server

Der Continuous-Integration-Server stellt bei der Umsetzung die maßgebliche Instanz dar. Durch Quellcode-Änderungen in der Versionsverwaltung wird auf diesem Server ein automatischer Build des gesamten Projektes ausgelöst. Die hierfür genutzte Umgebung wird nicht durch entwicklerspezifische Konfigurationen des Arbeitsplatzes beeinflusst. Das Projekt kann durch dieses Vorgehen auf einfache Weise auf die Reproduzierbarkeit von Ergebnissen überprüft und somit eine Aussage über die Unabhängigkeit von der Entwicklungsumgebung getroffen werden. Abbildung Abbildung 2 zeigt die Konfiguration des Build-Jobs in der Jenkins-Oberfläche, dem von uns verwendeten Continuous-Integration-Server.

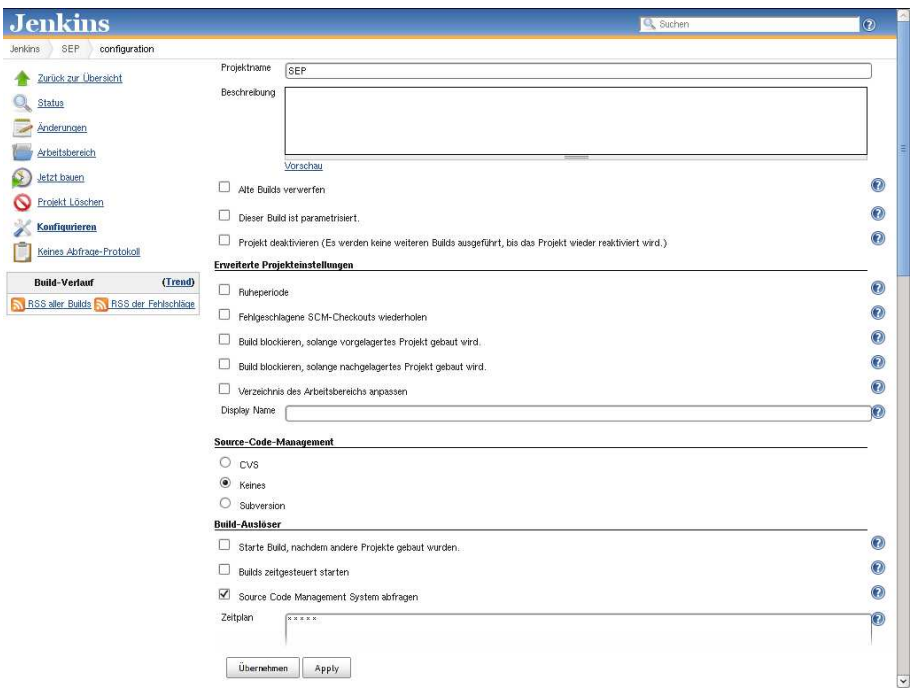

Abbildung 2: Build-Job Konfiguration in der Jenkins Oberfläche

In der Software-Entwicklung mit Java hat sich *Jenkins* als erfolgreiches Projekt[2] mit einer großen und aktiven Community[3] etabliert. Durch die aktive Community und der guten Erweiterbarkeit zeichnet sich Jenkins besonders durch eine Abdeckung vieler Automatisierungswerkzeuge aus. So unterstützt es derzeit gängige Build-Automatisierungen mittels Apache Ant und Maven sowie dem recht neuen und noch nicht so verbreiteten Gradle.

Jenkins wird als Java-EE-Projekt entwickelt und eignet sich zum Deployment auf einem

[2] Als Nutzer sei hier die JBoss-Community angeführt `http://hudson.jboss.org/hudson`

[3] Der Großteil der Community ist dem Haupt-Entwickler nach dem Fork von *Hudson* gefolgt

Application Server. Des weiteren wird es mit einem integriertem Servlet-Container ausgeliefert, was eine Standalone-Ausführung ermöglicht. Dementsprechend einfach gestaltet sich die Installation. Nach dem Start der Anwendung sind alle Konfigurationen über eine übersichtliche Web-Oberfläche möglich. Über diese Oberfläche kann auch eine Liste der durch die Community entwickelten Erweiterungen eingesehen und über einen einfachen Klick installiert werden.

Die Konfiguration der einzelnen Build-Jobs und Berechtigungen auf diesen erfolgt komfortabel über die Web-Oberfläche. Neben den Werkzeugen zur Build-Automatisierung und der Adresse des Quellcode-Repositories lassen sich für jeden Job gesondert Benachrichtigungseinstellungen über Fehlermeldungen bei der Ausführung anlegen. Über diesen Mechanismus wird der Benutzer, der die letzte Änderung vor dem fehlgeschlagenem Build in das Repository übertragen hat, auf den verursachten Fehler aufmerksam gemacht.

Neben der reinen Rolle als unabhängige Instanz zur Beurteilung des Projektstatus bietet Jenkins viele weitere Features, die es wert sind, in die Unternehmen getragen zu werden[4]. Sogenannte Matrix-Builds erlauben parametrisierbare Builds in Kreuzkonfigurationen gegen bestimmte Parameterwerte durchzuführen. Gerade in der starren Unternehmenswelt, in der Projekte gegen eine genau spezifizierte Laufzeitumgebung implementiert werden, ist eine Migration der Laufzeitumgebung zu von außen vorgegebenen Zeitpunkten unerlässlich. Mithilfe dieser Mehrfachkonfiguration kann ein Projekt frühzeitig auf Kompatibilität mit aktuellen und zukünftigen Bibliotheks- und Komponentenversionen getestet werden. Eine termingerechte, problemlose Umstellung der Migration kann somit schon in der aktuellen Entwicklung sichergestellt werden.

Zur Interaktion mit dem Quellcode-Repository bietet Jenkins mehrere Möglichkeiten. In der Web-Oberfläche kann die Konfiguration des jeweiligen Build-Jobs mittels eines an die Cron-Syntax angelehnten Ausdrucks angewiesen werden, das Repository in Intervallen nach Änderungen abzufragen. Wird die Infrastruktur insgesamt von vielen Projekten genutzt, erzeugt dieser Polling-Ansatz zusätzliche Last auf allen beteiligten Systemen. In einer solchen Situation sollte ein Build durch Repository-Hooks – Skripte, die bei Ereignissen durch den Repository-Server ausgeführt werden – ausgelöst werden. Jenkins bietet zu diesem Zweck eine einfache REST-Schnittstelle, mit deren Hilfe eine Aktion durch Fremdsysteme ausgelöst werden kann.

Durch die Erweiterbarkeit lässt sich Jenkins sehr einfach in bestehende Infrastrukturen zur Authentifizierung und Autorisierung integrieren. Namentlich sind hier *Reverse Proxy Auth*[5] zur Delegation der Authentifizierung an einen HTTP-Proxy und das *LDAP Email Plugin*[6] zur Ermittlung der E-Mail-Adresse eines Benutzers aus seiner LDAP-User ID genannt.

[4] [Sma11] bietet eine hervorragende Einführung in die Möglichkeiten, die Jenkins als CI-Server bietet

[5] `https://wiki.jenkins-ci.org/display/JENKINS/Reverse+Proxy+Auth+Plugin`

[6] `https://wiki.jenkins-ci.org/display/JENKINS/LDAP+Email+Plugin`

2.4 Metrik-Dashboard

Neben der starken Kommunikation sämtlicher Projektbeteiligten setzten agile Entwicklungsprozesse auch auf eine schnellere und umfangreichere Information über den Projektstatus.

In den vergangenen Jahren erlangten Werkzeuge und Plugins für Entwicklungsumgebungen zur Überprüfung der Einhaltung von Codier-Richtlinien und Hilfestellung bei bekannten Fehlermustern mehr Aufmerksamkeit. Für Entwickler ergab sich hieraus unter anderem der Vorteil, dass bekannte Schwachstellen und Fehler in der Software leichter zu beheben und auch direkt vermeidbar geworden sind. Der Einsatz am Entwicklerarbeitsplatz berücksichtigt jedoch die Arbeitsplatzkonfiguration – in der manche Konfigurationen sich erheblich von Produktivsystemen unterscheiden – und stellen darüber hinaus nur eine Momentaufnahme der Bewertung dar.

Das Metrik-Dashboard *Sonar*, dessen Projektübersichtsseite in Abbildung Abbildung 3 dargestellt ist, setzt genau an diesem Punkt an. Wie auch Jenkins erfordert Sonar die Ausführung auf einem Java-Application-Server. Zusätzlich wird eine Datenbank zur Speicherung der gesammelten Metriken benötigt. Nach erfolgter Installation kann Sonar über die Web-Oberfläche konfiguriert und benutzt werden.

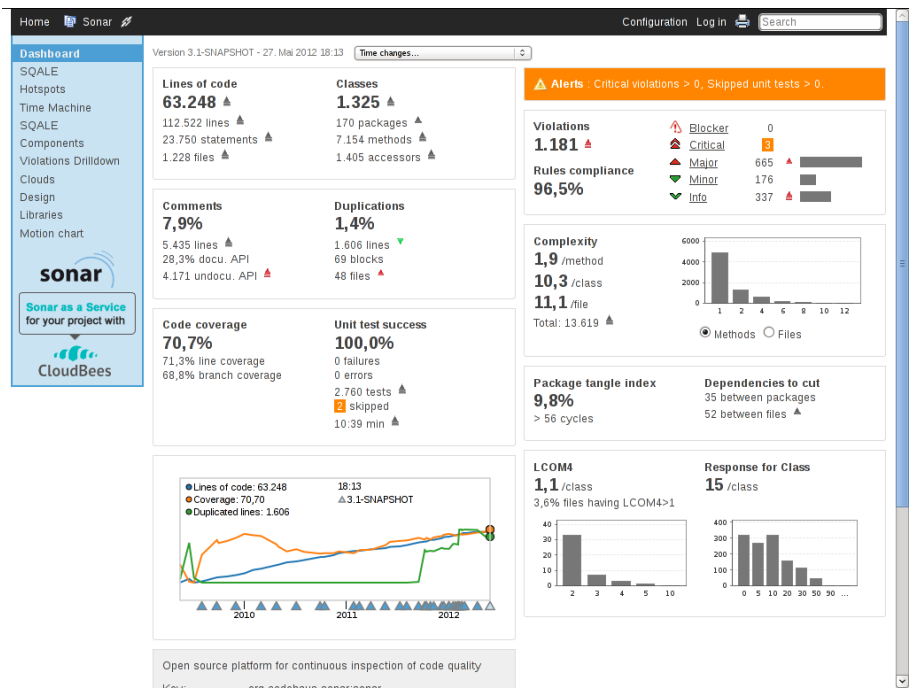

Abbildung 3: Projektübersicht innerhalb der Sonar-Oberfläche

Die eigentliche Metrik-Erhebung und Speicherung in der Datenbank führt – je nach Werkzeug zur Build-Automatisierung – ein Plugin während des Builds durch. In Verbindung mit Maven und Jenkins ermöglicht das Jenkins-Plugin für Sonar[7] eine automatische Metrik-Erhebung ohne explizite Konfiguration innerhalb der einzelnen Projekte. Für jedes Projekt wird übersichtlich aufgeführt, in welchen Quellcode-Bereichen Best-Practices oder Formatierungskonventionen missachtet worden sind. Diese sind auf vielfältige Weise navigierbar. Es kann nach Regeln und Schwere der Verstöße im gesamten Quellcode navigiert werden.

Aus Erfahrungswerten treten Mängel im Quellcode meist gehäuft in einzelnen Bereichen der Software auf. Sonar unterstützt die Suche nach solchen Teilen mit der Hot-Spot Funktion. Hierbei werden die Klassen mit der höchsten Fehlerdichte aufgelistet.

Sollten Regelverstöße im Projekt unumgänglich gewesen sein, können die Entwickler ähnlich wie in einem Bug-Tracker System markierte Stellen kommentieren, diskutieren und im Falle eines berechtigten Verstoßes für nachfolgende Erhebungen als false-positive markieren.

3 Integration in bestehende Infrastruktur

Zur Absicherung und einheitlichen Integration in bestehende Infrastrukturen bietet sich die Einrichtung eines HTTP-Daemons als vorgelagerter Proxy an. Dies ermöglicht nicht nur die Absicherung der nachgelagerten Systeme durch einheitliche SSL-Verschlüsselung und Autorisierung gegen ein zentrales LDAP-Verzeichnis, sondern vereinheitlicht auch den Zugriff auf die Dienste.

Sämtliche Services sind somit unter den bekannten HTTP/S Ports im Netzwerk erreichbar. Um einzelne Teilsysteme ansprechbar zu machen, bietet sich die Konfiguration von Sub-Domains oder Sub-Locations an. Somit wäre beispielsweise das Subversion-Repository und svn.server.net, der Build-Server unter jenkins.server.net und das Metrik-Dashboard unter sonar.server.net erreichbar.

Durch die zentrale Konfiguration der Authentifizierung und Autorisierung an den vorgelagerten HTTP-Server ist auch die Realisierung eines Single-Sign-On-Verfahrens einfach zu. Der einzelne Nutzer muss sich somit nur einmal pro Session am Server anmelden, hat jedoch Zugriff auf jeden Dienst und wird von diesem identifiziert.

4 Fazit

Die hier vorgestellten Systeme finden in vielen Unternehmen bereits Verwendung oder werden im Moment für den Einsatz evaluiert.

Subversion stellt als de facto Standard in vielen Bereichen eine ausgereifte und auch hin-

[7] `http://docs.codehaus.org/display/SONAR/Hudson+and+Jenkins+Plugin`

reichend bekannte Lösung dar. Neben der grundsätzlichen Verwendung stellt der Gebrauch jedoch auch Anforderungen hinsichtlich verbreiteter Konventionen. Im Lehrbetrieb können Studenten unter realen Bedingungen die Vor- und Nachteile erkennen und erlernen.

Durch Jenkins werden nicht nur die grundlegenden Anforderungen an einen Continuous-Integration-Server erfüllt. Durch den Ursprung im Open-Source-Bereich bietet er eine ausgefeilte Anpassung der Projekt-Builds und eine frühe Unterstützung von neuen Werkzeugen. Gerade die Möglichkeit der Matrix-Builds zur Sicherung der Migrationsfähigkeit eines Projekts findet in der Auftragsentwicklung bisher seltener Verwendung. Tragen Absolventen dieses Wissen in die Arbeitswelt, stellt dieses einen deutlichen Gewinn für alle Seiten dar.

Im Bereich des Software-Engineerings fallen häufig die Schlagworte des Refactorings, der Quellcode-Analyse und des Personal Software Processes. Sonar unterstützt Software-Entwickler durch eine Vielfalt an Metriken und Quellcode-Anmerkungen im richtigen und effektiven Umgang mit Programmiersprachen. Durch die Hot-Spot Analyse werden gezielt anfällige Projektteile auffindbar gemacht und können so zum Refactoring ausgewählt werden. Bekannte Fehlermuster und unsaubere Programmierung können direkt im Quellcode angezeigt und durch Erklärung den Grund und Effekt des Fehlers eingesehen werden.

Durch die Nutzung von aktiv weiterentwickelter Open-Source-Software entstehen durch die Nutzung keine Kosten, was für den Einsatz in der Hochschulausbildung eine gewichtige Rolle spielt. Die Installation der einzelnen Systeme ist selbst auf Consumer-Hardware ohne weitere Probleme möglich, da keine hohen Ressourcenanforderungen gestellt werden. Installation und Konfiguration können von Personal mit ein wenig Erfahrung in kurzer Zeit durchgeführt und angepasst werden.

Die ersten Erfahrungen in der Verwendung der genannten Infrastruktur im Sommersemester 2012 sind auf konzeptioneller und didaktischer Ebene vielversprechend. Die oben genannte einfache Installation und Konfiguration der einzelnen Systeme relativiert sich jedoch bei der Integration in die Infrastruktur eines (Hochschul-)Rechenzentrums. Hier ist ein nicht unerheblicher Aufwand zu investieren.

Literatur

[PSCM08] B. Porter, C. Sanches, J. Casey und V. Massol. *Better Builds With Maven*. Library Press, 2008.

[Sma11] J.F. Smart. *Jenkins: The Definitive Guide*. O'Reilly Media, 2011.