

Hochschule RheinMain
Fachbereich Design Informatik Medien
Studiengang Allgemeine Informatik

Diplomarbeit
zur Erlangung des akademischen Grades
Diplom-Informatiker (FH)

Erhöhung der Softwarequalität in kleinen Unternehmen durch einen Continuous Integration-Prozess

Vorgelegt von: Martin Rödiger
Vorgelegt am: 11.01.2010
Referent: Prof. Dr. Sven Eric Panitz
Korreferent: Dipl.-Inf. Martin Klossek
Betreuer: Dipl.-Inf. (FH) Patric Eid

Erklärungen:

Erklärung gemäß Prüfungsordnung – Teil A - §6.4.2

Ich versichere, dass ich die Diplomarbeit selbstständig verfasst und keine anderen als die angegebenen Hilfsmittel benutzt habe.

Ort, Datum

Unterschrift Diplomand

Hiermit erkläre ich mein Einverständnis mit den im Folgenden aufgeführten Verbreitungsformen dieser Diplomarbeit:

Verbreitungsform	ja	nein
Einstellung der Arbeit in die Bibliothek der FHW		X
Veröffentlichung des Titels der Arbeit im Internet	X	
Veröffentlichung der Arbeit im Internet		X

Ort, Datum

Unterschrift Diplomand

Vorwort

Ich möchte mich ganz besonders bei meinem Betreuer Patric Eid für die Unterstützung bei der Organisation der Arbeit und Sammlung von Informationen bedanken. Vor allem in den letzten Monaten der Arbeit war seine Unterstützung bei der Korrektur der Dokumentation sehr hilfreich.

Ich danke der Firma eWorks GmbH und insbesondere Martin Klossek für das interessante Thema und die Bereitstellung eines Notebooks und eines Arbeitsplatzes zur Durchführung der Diplomarbeit.

Mein Dank geht auch an meine Eltern und Freunde, insbesondere Martin Dommermuth und Simon Pietsch, die mir bei der Korrekturlesung der Arbeit behilflich waren.

Inhaltsverzeichnis

1	Einleitung.....	1
1.1	Aufbau der Arbeit.....	1
1.2	Einführung in Continuous Integration.....	1
1.3	Grundbausteine eines Continuous Integration Prozesses.....	2
1.3.1	Versionsverwaltung.....	2
1.3.2	Automatisierter Build-Prozess.....	3
1.3.3	Tests.....	4
1.3.4	Metriken.....	5
1.3.5	Continuous Integration Server.....	6
1.4	Problemstellung.....	6
1.5	Ziele der Arbeit.....	7
2	Evaluation von Continuous Integration Servern.....	9
2.1	Auswahl.....	9
2.1.1	Vorstellung der CIS.....	10
2.1.1.1	CruiseControl.....	10
2.1.1.2	CruiseControl.NET.....	10
2.1.1.3	Cruise.....	11
2.1.1.4	Anthill.....	11
2.1.1.5	Anthill Pro.....	11
2.1.1.6	Buildbot.....	12
2.1.1.7	Continuum.....	12
2.1.1.8	Hudson.....	13
2.1.1.9	CI Factory.....	13
2.1.1.10	Bamboo.....	14
2.1.1.11	Build Beat.....	14
2.1.1.12	OpenMake Meister.....	14
2.1.1.13	OpenMake Mojo.....	15
2.1.1.14	FinalBuilder Server.....	15
2.1.1.15	Pulse.....	16
2.1.1.16	TeamCity.....	16
2.2	Kriterien.....	16
2.2.1	Code.....	17
2.2.2	Versionsverwaltung.....	18
2.2.3	Build-Ausführung.....	20
2.2.4	Build-Auslöser.....	21
2.2.5	Zugriffsschutz.....	21
2.2.6	Veröffentlichung.....	21
2.2.7	Web-Oberfläche.....	22
2.2.8	GUI.....	22
2.2.9	CLI.....	23
2.2.10	Darstellung von Test-Ergebnissen.....	23
2.2.11	Direkter Build-Tool Support.....	23
2.2.12	Direkter Test-Framework Support.....	24
2.3	Gewichtung.....	24
2.4	Ergebnisse.....	27
2.5	Lizenzkosten.....	35
2.6	Direktvergleich der Favoriten.....	36
2.6.1	Bamboo.....	36
2.6.2	Hudson.....	39
2.6.3	Zusammenfassung.....	41

2.7 Eigenentwicklung.....	43
3 Erweiterung des Continuous Integration Servers.....	45
3.1 Plugin-Kategorien.....	46
3.1.1 Source code management.....	46
3.1.2 Build-Triggers.....	46
3.1.3 Build-Tools.....	47
3.1.4 Build-Wrappers.....	48
3.1.5 Build-Notifiers.....	49
3.1.6 Build-Reports.....	50
3.1.7 Artifact-Uploaders.....	56
3.1.8 External site/tool integrations.....	57
3.1.9 UI plugins.....	58
3.1.10 Authentication and user management.....	59
3.1.11 Verschiedenes.....	59
3.2 Verwaltung der Plugins.....	61
3.3 Plugin-Dateistruktur (HPI).....	64
4 Integration des Continuous Integration Servers.....	65
4.1 Aufbau des virtuellen Rechners.....	65
4.1.1 Hudson Installation.....	66
4.1.2 Tomcat Konfiguration.....	67
4.1.2.1 Umstellung der Hudson URL.....	67
4.1.2.2 Konfiguration der JVM.....	67
4.1.2.3 Windows Service Einstellungen.....	69
4.2 Integration der Projekte.....	69
4.2.1 Projektauswahl.....	69
4.2.2 Allgemeines zur Umsetzung in Hudson.....	70
4.2.2.1 Jobs.....	70
4.2.2.1.2 Verwendete Plugins.....	71
4.2.2.1.3 Konfiguration von Hudson.....	71
4.2.3 C#-Projekt.....	72
4.2.3.1 Ausgangslage.....	72
4.2.3.2 Umsetzung in Hudson.....	73
4.2.4 VB.NET-Projekt.....	77
4.2.4.1 Ausgangslage.....	78
4.2.4.2 Umsetzung in Hudson.....	78
4.2.4.2.1 Haupt-Job.....	78
4.2.4.2.2 Web-Job.....	82
4.2.4.2.3 Test-Server-Job.....	83
4.2.5 PHP-Projekt.....	84
4.2.5.1 Ausgangslage.....	85
4.2.5.2 Umsetzung in Hudson.....	85
5 Bewertung des CIS-Einsatzes.....	89
5.1 Projekte.....	90
5.1.1 C#-Projekt.....	90
5.1.2 VB.NET-Projekt.....	93
5.1.3 PHP-Projekt.....	100
6 Zusammenfassung.....	103
7 Abkürzungen.....	105
8 Glossar.....	107
9 Literaturverweise.....	109
10 Quellen.....	111
11 Abbildungsverzeichnis.....	113

12 Weitere Verzeichnisse.....	115
13 Anhang.....	117
13.1 Ergebnisse der Evaluation.....	117
13.2 Build-Skripte.....	121
13.2.1 C#-Projekt.....	122
13.2.2 VB.Net-Projekt.....	124
13.2.2.1 Haupt-Job.....	124
13.2.2.2 Web-Job.....	126
13.2.2.3 Test-Server-Job.....	127
13.2.3 PHP-Projekt.....	128

1 Einleitung

1.1 *Aufbau der Arbeit*

Die Arbeit ist wie folgt aufgebaut:

Zunächst werden in dem ersten Kapitel die Grundlagen des Themas erläutert. Dabei wird eine kurze Einführung in die kontinuierliche Integration von Software gegeben und die Bestandteile eines Continuous Integration Prozesses erläutert. Danach folgen die Aufgabenstellung und die Ziele der Arbeit.

Darauf hin folgt die Evaluation der Continuous Integration Server (CIS), an dessen Ende die Entscheidung für ein System oder eine Eigenentwicklung steht.

Nach der Entscheidung folgt die Erläuterung von Erweiterungsmöglichkeiten des CIS Hudson. Dabei wird besonders auf die Erweiterungen eingegangen, die bei der Integration von Projekten in den CIS benutzt wurden.

Danach folgt die Integration des CIS in der Firma eWorks GmbH und die Umsetzung ausgewählter Projekte.

Zum Abschluss wird der Einsatz von Hudson für diese Projekte bewertet.

1.2 *Einführung in Continuous Integration*

In der Software-Entwicklung werden bis zu der Veröffentlichung einer Anwendung viele Prozessschritte durchlaufen. Bei Projektteams mit mehreren Software-Entwicklern, arbeiten die Entwickler üblicherweise in einer auf ihren Rechnern eingerichteten Entwicklungsumgebung, damit sie sich nicht gegenseitig behindern. Der gemeinsam entwickelte Code wird in einer Versionsverwaltung gespeichert. Jeder Entwickler arbeitet lokal an seiner Kopie des Quellcodes, welcher aus der Versionsverwaltung heruntergeladen wird. Dieser Vorgang wird auch als „aus-checken“ oder „checkout“ bezeichnet. Damit Fehler frühzeitig abgefangen werden können, entwickeln geübte Entwickler neben dem eigentlichen Quellcode auch Test-Code. Dieser, gepackt in Testfällen und Test-Klassen, wird für gezielte Tests von Programmabschnitten eingesetzt. Der fachliche Ausdruck für diese Art von Testen lautet „UnitTest“. Für spezielle (und zumeist umfangreiche) Tests wird die Anwendung

1.2 Einführung in Continuous Integration

oftmals zusätzlich in einer Testumgebung geprüft. Im Idealfall enthält diese Umgebung, im Gegensatz zur Entwicklungsumgebung, außer dem Betriebssystem meist nur absolut notwendige Programme. Mit Hilfe solcher Testumgebungen lassen sich z.B. die Verhältnisse bei einem Kunden simulieren und Integrationsprobleme können rechtzeitig vor der Veröffentlichung der Anwendung erkannt (und natürlich behoben) werden. Die Ergebnisse der automatischen Tests werden üblicherweise allen Beteiligten in Dateiform zugänglich gemacht. Mit den Tests wird die korrekte Funktion einer Anwendung geprüft, durch die Ausführung sogenannter Metriken erhalten die Entwickler zusätzlich Information zu der Qualität der Anwendung.

Durch einen Continuous Integration-Prozess (deutsch: „Kontinuierliche Integration“) wird eine Anwendung regelmäßig automatisch neu gebaut, getestet und dessen Qualität überprüft. Dadurch erhalten die Entwickler eine schnelle Rückmeldung zu ihren Änderungen an der Anwendung. Die Testergebnisse werden bei jedem Bauen gesammelt und ermöglichen es auch für Nicht-Entwickler die Qualität der Anwendung über den Projektzeitraum zu verfolgen. Die Automatisierung verringert den Zeitaufwand der einzelnen Entwickler, neue Versionen einer Anwendung zu veröffentlichen. Durch die Sammlung der Testergebnisse, kann man einfach feststellen welche Version einer Anwendung den Qualitätsansprüchen genügt und an den Kunden weitergegeben werden kann. Sogar die Veröffentlichung einer Anwendung ließe sich mit einem CI-Prozess automatisieren.

1.3 Grundbausteine eines Continuous Integration Prozesses

Für die Etablierung eines kompletten CI Prozesses sollten die folgenden Bedingungen erfüllt werden.

1.3.1 Versionsverwaltung

Eine der Grundvoraussetzungen für einen CI Prozess ist eine gemeinsame Quellcodebasis für die Entwickler der Anwendung. Dies wird durch den Einsatz einer Versionsverwaltung erreicht. Für die Versionsverwaltung stehen viele Programme zur Verfügung. Im weiteren Verlauf dieser Diplomarbeit wird vermehrt auf die Versionsverwaltung Subversion¹ Bezug genommen.

¹ Subversion: <http://subversion.tigris.org>

Durch eine Versionsverwaltung wird der Quellcode archiviert und Änderungen durch die Entwickler erfasst. Eine neue Version des Codes löst den zugehörigen CI -Prozess aus.

1.3.2 Automatisierter Build-Prozess

Entwickler benutzen meistens eine grafische Entwicklungsumgebung, um den Quellcode zu bearbeiten und die Anwendung neu zu erstellen. Für einen CI-Prozess ist es jedoch erforderlich, dass der gesamte Erstellungsprozess automatisierbar ist. Zu diesem Zweck besitzen viele grafische Entwicklungsumgebungen auch eine Schnittstelle für Kommandozeilenbefehle an oder verwenden einen Compiler, der sich auch ohne die grafische Oberfläche ansteuern lässt.

Für die automatische Erstellung von Anwendungen bieten sich außerdem darauf spezialisierte „Build-Tools“ an. Häufig werden diese auch direkt von der grafischen Entwicklungsumgebung dazu verwendet. Weit verbreitete „Build-Tools“ sind z.B. Ant², Nant³, Maven⁴ und MSBuild⁵.

Bei den genannten „Build-Tools“ wird der Build-Prozess mit einer auf XML basierenden Konfigurationsdatei definiert. Wie eine solche XML-Datei aussehen kann, zeigt das folgende Beispiel des „Build-Tools“ Ant:

```
[1] <project>
[2]     <target name="clean">
[3]         <delete dir="build"/>
[4]     </target>
[5]     <target name="compile">
[6]         <mkdir dir="build/classes"/>
[7]         <javac srcdir="src" destdir="build/classes"/>
[8]     </target>
[9]     <target name="jar">
[10]        <mkdir dir="build/jar"/>
[11]        <jar destfile="build/jar/HelloWorld.jar"
basedir="build/classes">
[12]            <manifest>
```

2 <http://ant.apache.org/>

3 <http://nant.sourceforge.net/>

4 <http://maven.apache.org/>

5 <http://msdn.microsoft.com/de-de/library/ms171452%28VS.80%29.aspx>

1.3.2 Automatisierter Build-Prozess

```
[13]         <attribute name="Main-Class"  
value="oata.HelloWorld"/>  
[14]         </manifest>  
[15]     </jar>  
[16] </target>  
[17] <target name="run">  
[18]     <java jar="build/jar/HelloWorld.jar" fork="true"/>  
[19] </target>  
[20] </project>
```

Typische Aufgaben eines Build-Prozesses sind dabei:

- Konfiguration der Umgebung.
- Löschen alter Build-Artefakte.
- Kompilieren der Anwendung.
- Kompilieren von Testprogrammen.
- Durchführen der Testprogramme.
- Statische Code-Überprüfung durch Metrik Tools.
- Generierung von Dokumentation.

Für den Build-Prozess sollte ein eigener Rechner bereitgestellt werden, auf dem neben dem Betriebssystem nur die für den Build-Prozess notwendigen Hilfs-Programme installiert sind. Dadurch können häufige Integrationsprobleme, wie z.B. fehlende Bibliotheken, schnell durch das CI-System aufgedeckt und das Entwickler-Team darüber informiert werden.

1.3.3 Tests

Das erfolgreiche Kompilieren einer Anwendung garantiert nicht, dass diese im Betrieb wie vorgesehen funktioniert. Deshalb gehört zu einem vollständigen CI-Prozess die Ausführung von Testprogrammen, die das Verhalten der Anwendung überprüfen.

Abhängig von der Programmiersprache existieren verschiedene Testframeworks, die zum Testen der Funktionalitäten einer Anwendung eingesetzt werden können. Zu den bekanntesten gehören z.B. Junit für Java und NUnit für .NET Programme.

Je nach Komplexität der Tests und Entwicklungsstand der Anwendung unterscheidet man verschiedene Teststufen. Komponententests bilden dabei die niedrigste Stufe und prüfen die Funktionen einzelner Module, wie Unterprogramme oder Klassen.

Integrationstests testen die Zusammenarbeit einzelner Komponenten. Bei Systemtests wird die gesamte Anwendung gegen die Anforderungen getestet. Dabei wird in einer Testumgebung die Produktivumgebung des Kunden simuliert. Der Abnahmetest (oder auch Akzeptanztest) bildet dabei die höchste Stufe und wird durch den Kunden ausgeführt. Dieser hat dadurch die Möglichkeit den aktuellen Stand der Anwendung zu kontrollieren und in seinem Umfeld zu testen.

Bei einer Anwendung mit vielen automatischen Tests, nehmen diese in einem CI-Prozess die meiste Zeit in Anspruch. Damit die Entwickler zeitnah zu ihren Änderungen über Probleme unterrichtet werden können, sollten die Testzyklen möglichst kurz gehalten werden. Dazu ist es sinnvoll die automatischen Tests nach ihrer Laufdauer zu sortieren und die kurzen Tests zuerst auszuführen. Der gesamte CI Prozess, der bei einer Änderung an dem Code der Anwendung ausgelöst wird, sollte eine Laufdauer von 10 Minuten nicht übersteigen, damit die Entwickler zeitnah über die Resultate unterrichtet werden können. Besonders bei zeitintensiven und komplexen Systemtests ist es ratsam, diese in eigene Prozesse auszulagern und gesondert in größeren Zeitabständen auszuführen.

1.3.4 Metriken

Zu einem CI-Prozess gehört auch die Ausführung von Metrik-Tests, die die Qualität des Quellcodes beurteilen sollen.

Zur Beurteilung der Softwarequalität können verschiedene Kennzahlen herangezogen werden:

- Das Verhältnis zwischen Anzahl der Codezeilen und der Dokumentation
- Komplexität von Funktionen
- Abhängigkeiten zwischen den Klassen
- Code-Abdeckung der Testprogramme
- Suche nach kopierten Code-Abschnitten

Weitere für die Qualität interessante Informationen wären:

- Untersuchung der Quellcodes nach Sicherheitslücken
- Vergleich des Quellcodes mit festgelegten Konventionen (z.B. Einrückung, Klassen- und Methodennamen)

•Hinweise zum Programmierstil

Für alle diese Punkte gibt es mittlerweile eine große Anzahl von Programmen, die in einem CI-Prozess eingesetzt werden können.

1.3.5 Continuous Integration Server

Ein Continuous Integration Server sammelt für alle verwalteten Projekte die Ergebnisse der Builds und dessen Testdaten an einer zentralen Stelle und ermöglicht es auch Nicht-Entwicklern den Überblick über den Stand der Anwendungen eines Unternehmens zu behalten. Meistens werden die Informationen dabei über eine Weboberfläche zur Verfügung gestellt.

Bei einem fehlerhaften Build können die betroffenen Entwickler des Projekts benachrichtigt werden und gemeinsam eine Lösung für das Problem suchen. Durch die zeitnahe Benachrichtigung werden Fehler schneller behoben und der aktuelle Stand in der Versionsverwaltung ist immer lauffähig, besteht die Tests und erfüllt Anforderungen an die Qualität.

Die erfolgreich erstellten Versionen der Anwendungen könnten mit Hilfe eines CIS bereitgestellt werden und stehen so beispielsweise den Software-Testern oder Kunden jederzeit zur Verfügung.

Durch die Ausführung der Build-Prozesse auf einem CIS wird außerdem das Wissen zum Bauen der Anwendungen dokumentiert und zentral gesammelt. Dadurch wird es nicht zu einem Problem, wenn der normalerweise für die Builds zuständige Entwickler nicht mehr verfügbar ist.

Die Ausführung der Builds auf einem dedizierten CIS erleichtert außerdem die Arbeit der Entwickler, weil Sie ihre eigenen Arbeitsrechner weniger belasten müssen und einfach mit der Entwicklung der Anwendung fortfahren können.

1.4 Problemstellung

In der Firma eWorks GmbH wurden bisher in nur wenigen größeren Projekten automatisierte Build-Vorgänge eingesetzt. Die Ausführung von Testprogrammen und Metriken in den Projekten war oft ein von den Entwicklern ausgeführter manueller Prozess.

Als ein kleineres Unternehmen mit etwa 15 Mitarbeitern stellt sich für eWorks das Problem, dass es durch die relativ geringe Anzahl an Mitarbeitern kaum möglich ist einzelne Personen ausschließlich für die Administration eines CIS einzusetzen. Im Normalfall sind die Entwickler gleichzeitig auch Administratoren ihrer Projekte. Dadurch ist eine einfache Konfiguration und Verwaltung der Projekte in dem CIS ein wichtiger Punkt. Dabei spielt auch der Einarbeitungsaufwand der Mitarbeiter in das System eine Rolle. In kleineren Unternehmen werden häufig viele kleinere Projekte mit kurzer Projektdauer durchgeführt, deshalb sollte sich die Einrichtung der Projekte einfach gestalten und nicht zu lange dauern. Bei einem Projekt mit 2 Wochen Laufdauer lohnt es sich nicht, wenn man die Konfiguration des CIS mehr als einen Tag benötigt.

Durch den geringeren Umsatz kleiner Unternehmen spielen auch die Anschaffungskosten für den CIS eine größere Rolle. In diesem Zusammenhang ist in der Arbeit auch zu prüfen, ob sich eventuell auch eine Eigenentwicklung eines CIS lohnen würde.

1.5 Ziele der Arbeit

In dieser Arbeit soll überprüft werden, ob sich aktuelle CIS für den Einsatz in kleineren Unternehmen eignen und durch dessen Einsatz eine Verbesserung der Softwarequalität und die Einsparung von Arbeitszeit erreicht werden kann.

Im Vorfeld wird dazu eine Evaluation aktueller CIS durchgeführt. Mit Hilfe eines Kriterienkatalogs, der auf die Bedürfnisse des Unternehmens ausgerichtet ist, werden die CIS bewertet und eine Auswahl getroffen. Dabei wird auch die Möglichkeit einer Eigenentwicklung in Betracht gezogen.

Im Anschluss werden verschiedenen ausgewählten Projekten von eWorks in den CIS integriert und der Einsatz des CIS anhand der gesammelten Daten bewertet.

2 Evaluation von Continuous Integration Servern

Die Evaluation bildet die Grundlage für die Entscheidung, ob eine Eigenentwicklung oder die Verwendung (und ggf. Erweiterung) eines vorhandenen CIS sinnvoller ist. Mittlerweile gibt es ein sehr großes Angebot an kostenfreien und kommerziellen CIS auf dem Markt.

Um möglichst viele CIS in die Evaluation mit aufzunehmen, wurde die Evaluation in mehreren Phasen durchgeführt und dabei die Bewertung von Funktionsumfang und Benutzerfreundlichkeit getrennt.

In der ersten Phase wurden die einzelnen CIS anhand der angebotenen Funktionen bewertet. Für die Bewertung wurde eine Liste mit Kriterien erstellt, die die Eigenschaften und Funktionen umfasst, die für das Unternehmen am wichtigsten sind. Durch die Kombination von erfüllten Kriterien und deren Gewichtung entstanden die Punkte für die Bewertung.

In der zweiten Phase wurden die zwei CIS mit der höchsten Punktzahl für die finale Entscheidung installiert und im direkten Vergleich auf Handhabung der Benutzeroberfläche und Erweiterungsmöglichkeiten hin untersucht. Dabei wurde besonderen Wert auf die Benutzerfreundlichkeit bei der Einrichtung und Konfiguration von Projekten gelegt und die Bedienbarkeit und Darstellung der Oberfläche verglichen. Durch die genauere Untersuchung in dieser Phase ist es außerdem möglich, die Angaben aus der ersten Phase nochmal zu überprüfen. Bei größeren Missständen im Funktionsumfang oder in der Benutzerfreundlichkeit könnte man ggf. die Entscheidung zu Gunsten eines anderen CIS der ersten Phase treffen oder auf eine Eigenentwicklung zurückgreifen.

2.1 Auswahl

Für die Grundausswahl der zu testenden CIS wurden verschiedene Quellen herangezogen. Zum einen werden in dem Anhang des Buches „Continuous Integration“ von Paul M. Duvall verschiedene CI-Tools aufgeführt und kommentiert. Unter dem Begriff „Continuous Integration“ findet sich in der Internet-Enzyklopädie Wikipedia ebenfalls eine größere Liste⁶ von CIS. In diesen beiden Quellen wird außerdem auf eine

⁶ http://de.wikipedia.org/wiki/Continuous_Integration (Stand 01.12.2009)

2.1 Auswahl

größere Vergleichstabelle⁷ hingewiesen, in der eine Reihe bekannter CIS anhand ihre Funktionen verglichen werden. Bereitgestellt wird diese Tabelle auf den Wiki-Seiten des OpenSource CIS CruiseControl. Da die Informationen jedoch nicht immer korrekt sind, oder dem aktuellen Stand der Anwendungen entsprechen, sind die in der Vergleichstabelle gemachten Angaben mit Vorsicht zu genießen. Als eines der ersten und noch immer weit verbreiteten CIS wird CruiseControl bei Vergleichstests gerne als Referenz herangezogen und in dessen Wiki finden sich auch einige Artikel zu Alternativen⁸ CIS.

2.1.1 Vorstellung der CIS

In den folgenden Unterkapiteln werden die untersuchten CIS kurz vorgestellt. Als Anhaltspunkte für die Aktivität der Entwickler, wurden soweit möglich Daten über die Häufigkeit neuer Releases aufgeführt und die Instandhaltung der Homepage beurteilt.

2.1.1.1 CruiseControl

CruiseControl (kurz: CC) ist ein Java OpenSource Projekt und einer der ersten CIS auf dem Markt. Das erste Release 1.0 wurde im März 2001 veröffentlicht und ist seit Oktober 2009 in der Version 2.8.3 erhältlich. Seit Version 2.0 im September 2002 gab es etwa 3 Releases pro Jahr. Die Beiträge zum Quellcode des Projekts werden von insgesamt 9 Entwicklern eingepflegt, die die notwendigen Rechte für Änderungen am Quellcode in der Versionsverwaltung besitzen. Ursprünglich wurde CC von Thoughtworks Studios (<http://www.thoughtworks-studios.com/>) entwickelt und in ein OpenSource Projekt umgewandelt.

Die Homepage „<http://cruisecontrol.sourceforge.net/index.html>“ macht einen guten Eindruck. Die Releases werden unter „<http://sourceforge.net/projects/cruisecontrol>“ bereitgestellt und die Statistiken zeigen für die Releases aus 2009 ca. 68.600 Downloads an.

2.1.1.2 CruiseControl.NET

CruiseControl.NET (kurz: CC.NET) ist ein C# OpenSource Projekt nach dem Vorbild

⁷ <http://confluence.public.thoughtworks.org/display/CC/CI+Feature+Matrix> (Stand 01.12.2009)

⁸ <http://confluence.public.thoughtworks.org/display/CC/Understanding+the+alternatives+to+CruiseControl> (Stand 01.12.2009)

von CC. Das Release der Version 1.0 wurde August 2005 veröffentlicht. Seit dem wurden pro Jahr etwa 3 neue Releases veröffentlicht und die aktuellste Version 1.5.0 ist seit Oktober 2009 erhältlich.

Die Homepage „<http://confluence.public.thoughtworks.org/display/CCNET/>“ besteht aus einem Wiki-Forum und macht einen mittelmäßigen Eindruck. Die Installations-Dateien kann man sich unter „<http://sourceforge.net/projects/ccnet/>“ herunterladen und die Statistiken zeigen für die Releases aus 2009 etwa 57.000 Downloads an.

2.1.1.3 Cruise

Cruise ist ein kommerzieller CIS der Firma Thoughtworks Studios und wurde in Java entwickelt. Die Version 1.0 wurde im Juli 2008 veröffentlicht, seit dem gab es 5 weitere Releases. Die aktuellste Version 1.3.2 ist seit September 2009 verfügbar.

Laut den Entwicklern zeichnet sich der CIS durch die einfache Konfiguration von Agenten zur Durchführung verteilter Build-Prozesse aus. Die Gestaltung Weboberfläche ist außerdem so optimiert worden, dass der Status eines Releases über die 3 Phasen Build, Test und Veröffentlichung übersichtlich dargestellt wird. Dabei können die Veröffentlichungen auch gezielt manuell ausgeführt werden.

Die Homepage <http://www.thoughtworks-studios.com/> macht einen sehr guten Eindruck.

2.1.1.4 Anthill

Bei diesem CIS handelt es sich um den Vorgänger von Anthill Pro der Firma Urbancode (<http://www.urbancode.com/>). Anthill wird vom Hersteller als OpenSource Software zur Verfügung gestellt, allerdings sind auf der Hersteller-Homepage keine Verweise darauf zu finden. Mit etwas Hilfe von einer Internet-Suchmaschine wie www.google.de stößt man dennoch auf die Anthill Homepage mit der Adresse „<http://www.anthillpro.com/html/products/anthillos/default.html>“. Laut der den Angaben auf dieser Seite wurde die Version 1.0 im Juni 2001 veröffentlicht und seit dem letzte Release 1.8.1 im Februar 2006 nicht mehr weiter entwickelt.

2.1.1.5 Anthill Pro

Dieser kommerzielle CIS wird von der Firma Urbancode entwickelt. Das Release 3.1 wurde im Dezember 2006 veröffentlicht, seit dem gab es pro Jahr 3 bis 6 weitere

2.1.1.5 Anthill Pro

Releases bis zur aktuellsten Version 3.7.1 im Oktober 2009.

Die Entwickler legten in dem CIS besonderen Wert auf zusätzliche Funktionen, die die Weitergabe von Build-Artefakten an andere Abteilungen eines Unternehmens zur Qualitätssicherung, Durchführung von Benutzer-Akzeptanz-Tests und Einspielung in die Produktionsumgebung vereinfachen.

Die Homepage <http://www.anthillpro.com/> macht einen guten Eindruck.

2.1.1.6 Buildbot

Buildbot ist eine in Python geschriebene OpenSource Anwendung. Das erste Release erschien im April 2003 und ist seit August 2009 in der Version 0.7.11p3 erhältlich. Die 6 Releases aus 2009 wurden etwa 10.000-mal heruntergeladen.

Eigene Build-Vorgänge können in der Programmiersprache Python entwickelt werden. Die Konfiguration der verwalteten Projekte geschieht ebenfalls durch Python-Kommandos.

Die Homepage „<http://buildbot.net>“ macht einen etwas vernachlässigten Eindruck. Die Dokumentation wird unter „<http://djmitche.github.com/buildbot/docs/>“ gepflegt und der Quellcode und die Releases können unter „<http://sourceforge.net/projects/buildbot/>“ heruntergeladen werden.

2.1.1.7 Continuum

Dieser CIS ist ein OpenSource Projekt der Apache Software Foundation (<http://www.apache.org>) und wurde in Java entwickelt. Das erste Release wurde 2005 veröffentlicht. In 2009 gab es insgesamt 5 Releases und die neueste Version 1.3.4 (beta) wurde im September 2009 herausgebracht. Insgesamt gibt es 18 Entwickler aus verschiedenen Zeitzonen, die Änderungsrechte für den Quellcodes besitzen.

Der CIS ist auf die Verwendung der hauseigenen Build-Tools Ant, Maven 1 und 2 optimiert.

Die Homepage <http://continuum.apache.org/index.html> macht einen guten Eindruck.

2.1.1.8 Hudson

Dieser CIS wurde ursprünglich von einem Mitarbeiter der Firma Sun Microsystems⁹ in Java entwickelt und ist seit vielen Jahren ein OpenSource Projekt. Das erste Release erschien 2005 und die neueste Version 1.338 ist seit Dezember 2009 verfügbar. Die Zugriffsrechte für Änderungen an Hudsons Quellcode werden großzügig an die Mitglieder der Community vergeben. Das hat zur Folge, dass neue Releases in sehr kurzen Zeitabständen von 1-2 Wochen veröffentlicht werden.

Hudson zeichnet sich vor allem durch die flexible Konfiguration der verwalteten Projekte und seine hohe Erweiterbarkeit durch sogenannte Plugins aus. Die aktive Community hat mittlerweile über 200 Plugins veröffentlicht, die den Funktionsumfang von Hudson erweitern.

Im Rahmen einer Lizenz für die Entwicklungsplattform Glassfish¹⁰ bietet Sun Microsystems inzwischen auch eine kommerzielle Version von Hudson an. Dabei werden aus der öffentlichen OpenSource Variante etwa alle 6 Monate neue Releases erstellt. Dazwischen bereitgestellte Updates beseitigen nur Fehler und enthalten keine neuen Funktionen, die die Stabilität der Anwendung gefährden könnten.

Die Homepage „<http://hudson-ci.org/>“ macht einen guten Eindruck. Auf Hudsons Wiki-Seiten (<http://wiki.hudson-ci.org/display/HUDSON/Home>) werden außerdem die Dokumentationen zu den Plugins bereitgestellt.

2.1.1.9 CI Factory

Dieser CIS basiert auf CruiseControl.NET und erweitert diesen um eine eigene Weboberfläche und weitere zusätzliche Funktionen, die unter anderem die Konfiguration des CIS und der verwalteten Projekte vereinfachen sollen. Es handelt sich ebenfalls um ein OpenSource Projekt. Das erste Release mit der Version 0.7 wurde im Dezember 2006 veröffentlicht. Im Jahr 2009 wurden 2 Releases erstellt, wobei die aktuellste Version 1.1.0.47 (beta) seit August 2009 erhältlich ist.

Die Homepage <http://cifactory.org> macht einen mittelmäßigen Eindruck. Die Releases und der Quellcode werden unter <http://code.google.com/p/ci-factory/> bereitgestellt. Laut den Informationen auf dieser Seite, wird von insgesamt 6 Entwicklern gepflegt.

⁹ <http://www.sun.com/>

¹⁰ http://www.sun.com/software/products/glassfish_portfolio/get_it.jsp

2.1.1.10 Bamboo

Bamboo ist ein kommerzieller CIS der Firma Atlassian (<http://www.atlassian.com/>), welche unter anderem für die Anwendungen JIRA (System zur Dokumentation von Fehlermeldungen und Änderungen eigener Anwendungen) und Clover (Ermittlung der Codeabdeckung von UnitTests) bekannt ist. Die Integration dieser Anwendungen in Bamboo ist von den Entwicklern bereits vorgesehen und der Hersteller bietet dafür günstigere Lizenzen im Paket mit Bamboo an. Das erste Release der Version 1.0 wurde im Februar 2007 veröffentlicht, seit dem gab es jedes Jahr 2 bis 3 weitere Releases. Seit Januar 2010 ist die Version 2.5 erhältlich.

Bamboo zeichnet sich durch die benutzerfreundliche Verwaltung von Agenten zur Durchführung verteilter Build-Prozesse aus und bietet über die Weboberfläche viele Funktionen zur Erstellung von Statistiken über die verwalteten Projekte an.

Die Homepage <http://www.atlassian.com/software/bamboo/> macht einen sehr guten Eindruck und bietet unter anderem auch viele Videos zur Demonstration von Bamboo an. Zur Unterstützung der Plugin-Entwicklung besitzt Atlassian für seine Produkte eine weitere Internetseite unter <https://plugins.atlassian.com/search/by/bamboo>.

2.1.1.11 Build Beat

Hierbei handelt es sich um einen kommerziellen CIS der Firma Timpani Software (<http://www.timpanisoftware.com/>). Der Hersteller wirbt mit einer gegenüber von CC und CC.NET einfacheren Installation und Konfiguration des CIS und der verwalteten Projekte.

Die Homepage macht einen etwas vernachlässigten Eindruck. Leider ist es nicht ersichtlich wie oft neue Releases herausgebracht werden oder wie aktuell die neueste Version des CIS ist. Der letzte News-Eintrag auf der Hauptseite scheint von 2007 zu stammen.

2.1.1.12 OpenMake Meister

Dieser kommerzielle CIS wird vom Hersteller OpenMake Software (<http://www.openmakesoftware.com/>) entwickelt. Leider ist es nicht genau ersichtlich, wie oft von den Entwicklern neue Releases oder Updates herausgebracht werden. Die aktuellste Version 7.3 ist seit Juli 2009 erhältlich, davor wurde die Version 7.2 im Juli

2008 veröffentlicht.

Der CIS zeichnet sich durch eine besondere Funktion, genannt „Pre-Commit-Build“, aus. Damit können die einzelnen Entwickler ihre Änderungen am Quellcode auf dem CIS testen, bevor sie in die Versionsverwaltung aufgenommen werden. Grafische Entwicklungsumgebungen wie beispielsweise Visual Studio oder Eclipse werden durch Plugins um weitere Funktionen erweitert, über die sich die Build-Vorgänge der verwalteten Projekte konfigurieren lassen.

Die Homepage macht einen guten Eindruck und enthält viele Videos zur Demonstration der Anwendung.

2.1.1.13 OpenMake Mojo

Dieser CIS ist ebenfalls ein Produkt der Firma OpenMake Software. Es handelt sich dabei um eine „kleinere“ Variante von OpenMake Meister, bei der bestimmte Funktionen fehlen. Dazu gehören beispielsweise die bereits erwähnten Pre-Commit-Builds, automatische Generierung der Projektkonfiguration (aus Projekt-Dateien der Entwicklungsumgebung) und Beschleunigung der Kompilierung durch Multi-Threading. Die Releases erscheinen parallel zu denen von OpenMake Meister.

Es handelt sich um die gleiche Homepage wie bei OpenMake Meister.

2.1.1.14 FinalBuilder Server

Dieser kommerzielle CIS wird von der Firma VSoft Technologies (<http://www.finalbuilder.com/>) entwickelt. Der FinalBuilder Server in der Version 6 wurde das erste mal im April 2008 veröffentlicht. Im Jahr 2009 gab es 5 Releases und die aktuellste Version 6.3.0.1816 wurde im November 2009 zur Verfügung gestellt.

Der Hersteller bietet weitere grafische Anwendungen an, um die Arbeit mit dem CIS zu vereinfachen. Mit der FinalBuilder IDE lassen sich die Build-Vorgänge definieren und im CIS Server wie andere Build-Skripte zum bauen der Projekte verwenden. Dabei lassen sich die Build-Schritte aus vielen vordefinierten Aktionen zusammenstellen ohne dafür Build-Skripte schreiben zu müssen. Weitere Aktionen lassen sich mit Hilfe des Action Studios erstellen und der IDE hinzufügen.

Die Homepage <http://www.finalbuilder.com/> macht einen guten Eindruck. Von der

2.1.1.14 FinalBuilder Server

Community veröffentlichte Aktionen lassen sich unter <http://www.finalbuilder.com/community-downloads.aspx> herunterladen.

2.1.1.15 Pulse

Dieser kommerzielle CIS wird von der Firma Zutubi (<http://www.zutubi.com/>) entwickelt. Das erste Release der Version 1.0 wurde im August 2006 veröffentlicht. In 2009 gab es 3 Releases, wovon die aktuellste Version 2.1 im Dezember erschien.

Als besondere Funktion bietet Pulse genauso wie OpenMake Meister das Durchführen persönlicher Builds an, dabei können die Entwickler ihre Änderungen auf dem CIS testen, bevor sie in die Versionsverwaltung integriert werden. Die Konfiguration der verwalteten Projekte wird durch das Anlegen von Projekt-Gruppen vereinfacht. Teile der Projekt-Konfiguration können dabei in die Gruppen ausgelagert werden, was redundante Informationen in den Konfigurationen vermindert.

Die Homepage <http://www.zutubi.com/products/pulse/> hinterlässt einen guten Eindruck.

2.1.1.16 TeamCity

TeamCity ist ein kommerzieller CIS und wird von der Firma JetBrains (<http://www.jetbrains.com>) entwickelt. Die aktuellste Version 5.01 wurde im Dezember 2009 veröffentlicht. Im Jahr 2009 gab es insgesamt 9 Releases.

Der Hersteller bietet zu verschiedenen Entwicklungsumgebungen wie Visual Studio ,Eclipse und dem hauseigenen IntelliJ IDEA Plugins an. Damit können die Entwickler den eigenen Code auf dem CIS testen lassen, bevor sie ihn in die Versionsverwaltung übernehmen. Der CIS bietet eine weitere interessante Funktion zur Ausführung von UnitTests, dabei kann man einstellen, ob neue oder fehlgeschlagene Tests des letzten Builds als erstes ausgeführt werden sollen.

Die Homepage <http://www.jetbrains.com/teamcity/index.html> macht einen sehr guten Eindruck.

2.2 Kriterien

Der Kriterienkatalog für die Bewertung der CIS orientiert sich an den Bedürfnissen des Unternehmens und der Art der Projekte, die von den Entwicklern durchgeführt werden.

Werden in einem Unternehmen bevorzugt bestimmte Programmiersprachen und Entwicklungswerkzeuge für die Umsetzung der Projekte verwendet, sollte auch eine Unterstützung für diese Sprachen und Werkzeuge als Kriterium aufgenommen werden.

In der Firma eWorks GmbH werden die meisten Projekte mit .NET (C#, VBA, VB) oder PHP umgesetzt. Dabei verwenden die Entwickler meistens die Entwicklungsumgebung MS Visual Studio für .NET-Projekte oder Zend Studio für PHP-Projekte. Zur Versionsverwaltung des Quellcodes wird hauptsächlich die Software Subversion eingesetzt. In wenigen Fällen kommt auch noch CVS zum Einsatz. Ein sehr geringer Anteil der Projekte wurde in Java entwickelt. Insgesamt arbeiten bei eWorks etwa 15 Entwickler und als Betriebssystem für die Arbeitsrechner ist vorwiegend Windows Vista installiert.

Aufgrund dieser Tatsachen wurde insbesondere auf die Unterstützung von Build-Tools und Testframeworks für .NET, PHP und Java geachtet und als Kriterium aufgenommen.

Für kleinere Unternehmen ist es wahrscheinlich nicht rentabel einen Mitarbeiter speziell für die Administration und Pflege des CIS abzustellen. Aus diesem Grund gewinnt die einfache Konfiguration der Projekte und die Bedienung der Benutzeroberfläche an hoher Bedeutung.

Die einzelnen Kriterien wurden nach Themenbereichen gruppiert und nach ihrer Wichtigkeit für das Unternehmen gewichtet. In den folgenden Unterkapiteln werden die Themenbereiche und Kriterien erläutert. Die Gewichtungen befinden sich zur besseren Übersicht in einer eigenen Tabelle in Kapitel 24.

2.2.1 Code

Dieser Bereich fasst Kriterien zusammen, die den Quellcode und die Erweiterbarkeit des CIS betreffen.

- **Quellcode erhältlich**
Ist der Quellcode erhältlich, erleichtert dies die Erweiterung des CIS um weitere Funktionen oder die Anpassung der Oberfläche.
- **Redistribution erlaubt**
Ist eine Redistribution des CIS durch dessen Lizenz erlaubt, eröffnet dies eine weitere Einnahmequelle für das Unternehmen. So könnte z.B. eine eigene, erweiterte Version des CI-Servers an die Kunden lizenziert werden.
- **Plugin-Support**

Für die meisten Projekte sehen die Build-Prozesse, die von einem CIS ausgeführt werden, gleich aus. Der Quellcode wird kompiliert, es werden Tests ausgeführt und die Anwendung zur Veröffentlichung vorbereitet. Es kommt aber oft vor, dass für die Projekte zusätzliche, besondere Aufgaben zu erfüllen sind. Ein Beispiel wäre das Auslesen oder Modifizieren von Konfigurationsdateien einer Anwendung bevor sie an unterschiedliche Kunden weitergegeben werden kann. Der CIS sollte sich um weitere Funktionen erweitern lassen, um diese besonderen Aufgaben zu erfüllen und auch für zukünftige Projekte verfügbar zu machen.

2.2.2 Versionsverwaltung

Der CIS sollte in der Lage sein die Versionsverwaltung abzufragen und den aktuellsten Quellcode herunterzuladen. Es kann durchaus auch vorkommen, dass der Quellcode auf mehrere Repositories einer Versionsverwaltung verteilt ist. In dem Fall ist es interessant, ob der CIS die Überwachung mehrere Repositories pro Projekt erlaubt. Bisher kam dies es in den Projekten von eWorks jedoch noch nicht vor und das Problem könnte auch durch eine Aufteilung in mehrere Projekte gelöst werden. Daher ist dieses Kriterium als nicht so wichtig eingestuft worden. Unter Umständen werden in der Versionsverwaltung auch Dateien abgelegt, die vom CIS nicht überwacht werden sollen (z.B. Dokumentationen) oder man möchte nur bei Änderungen an bestimmten Dateien den CI-Prozess auslösen. Für dieses Problem ist es hilfreich, wenn man die Überwachung der Dateien filtern kann.

In der Firma eWorks GmbH wird fast ausschließlich Subversion zur Versionsverwaltung eingesetzt, deshalb besitzt die Unterstützung für Subversion eine sehr hohe Priorität. Nur der Quellcode mancher älterer Projekte befindet sich noch in einer CVS Versionsverwaltung, daher wurde eine Unterstützung für CVS mit einer sehr geringen Gewichtung in die Kriterien aufgenommen.

CVS wird inzwischen nicht mehr weiterentwickelt und Subversion versteht sich als dessen Nachfolger. Subversion ist eine „Freie Software“. Das bedeutet, dass der Quellcode frei verfügbar ist und für jeden Zweck genutzt, verändert und in ursprünglicher oder veränderter Form weiterverbreitet werden darf. Gegenüber CVS besitzt Subversion verschiedene Vorteile. SVN versioniert oder revisioniert bei Änderungen grundsätzlich das gesamte Projektarchiv und damit jeweils die gesamte Verzeichnisstruktur, während CVS auf der unabhängigen Versionierung jedes einzelnen Inhalts beruht. Dadurch ist es einfacher die exakte Version einer Datei zu beschreiben

(z.B. „Revision 123“ statt „Version vom 11.01.2009 12:00:00 GMT“). Werden von der Hauptentwicklungslinie des Quellcodes bestimmte Revisionen markiert (englisch: tagging) oder Abzweigungen (englisch: branching) erstellt, werden auf dem Subversion-Server Kopien aller betroffenen Dateien angelegt und ebenfalls mit einer Revisionsnummer versehen. So ist es z.B. sehr einfach eine bestimmte Version einer Abzweigung gezielt auszuchecken und zu bauen. Ändert ein Entwickler seinen ausgecheckten Quellcode, legt Subversion eine lokale Kopie der Dateien an, dadurch verdoppelt sich der Speicherbedarf. Allerdings hat dies auch einige Vorteile, so können die lokalen Änderungen ohne eine Anfrage an den Subversion-Server angezeigt und auch wieder rückgängig gemacht werden. Außerdem müssen beim Übertragen nur die geänderten Teile an den Server gesendet werden, während CVS die Änderungen Serverseitig berechnet und dazu die gesamte Datei benötigt. Die Übernahme von Änderungen in das Archiv ist ein atomarer Vorgang. Tritt mitten im Vorgang ein Fehler auf, werden alle Änderungen rückgängig gemacht und das Repository bleibt in einem konsistenten Zustand. Beim Kopieren von Dateien im Repository, ist Subversion in der Lage dessen gesamten Versionsverlauf ebenfalls zu übertragen. Dies ist auch beim Verschieben und Umbenennen von Dateien interessant, bei CVS führen diese Vorgänge zu einem Bruch im Versionsverlauf. Generell erlaubt Subversion im Gegensatz zu CVS auch die Verwaltung von Verzeichnissen und Metadaten. Insbesondere das Löschen von Verzeichnissen ist in CVS nicht ohne den Verlust der Historie aller enthaltenen Dateien möglich. Subversion selbst liefert keine grafische Benutzeroberfläche mit, mittlerweile gibt es aber ein großes Angebot an Clients¹¹. In eWorks wird die Benutzeroberfläche TortoiseSVN¹² eingesetzt. Diese integriert sich in den Windows-Explorer und ermöglicht einen besonders einfachen Umgang mit den versionierten Dateien. Kleine Symbole neben den Dateinamen machen sichtbar, ob die Dateien modifiziert wurden. Sämtliche Operationen wie das Aus- und Einchecken, Aktualisieren, Hinzufügen und Löschen von Dateien und selbst das Anlegen von Abzweigungen von der Hauptentwicklungslinie lassen sich mit wenigen Mausklicks durchführen. Außerdem ist es möglich die Änderungen im Repository nach zu verfolgen oder die Unterschiede zwischen zwei Versionen einer Datei anzuzeigen.

Subversion ist weit verbreitet und wird z.B. auch von vielen Anbietern kostenloser

¹¹ <http://subversion.tigris.org/links.html#all-clients>

¹² <http://tortoisesvn.tigris.org/>

2.2.2 Versionsverwaltung

Softwareverwaltung¹³ angeboten. In eWorks wird Subversion schon seit vielen Jahren zu Versionsverwaltung eingesetzt. Die Mitarbeiter sind dadurch gut mit Subversion und TortoiseSVN vertraut und haben bisher keine negativen Erfahrungen damit gemacht. Da in Zukunft kein Umstieg auf eine andere Software geplant ist, wurden keine weiteren Anbieter für die Bewertung berücksichtigt.

- **Subversion**
Der CIS unterstützt die Versionsverwaltung Subversion
- **CVS**
Der CIS unterstützt die Versionsverwaltung CVS
- **Multiple Repositories pro Projekt**
Unter Umständen befindet sich der Quellcode eines Projekts in mehreren Repositories. Erlaubt der CIS die Verwendung mehrerer Repositories pro Projekt, ist dieses Kriterium erfüllt.
- **Dateifilter**
Der CIS bietet die Möglichkeit an, bestimmte Dateien und Verzeichnisse in der Versionsverwaltung zu überwachen oder zu ignorieren.

2.2.3 Build-Ausführung

Dieser Bereich fasst die Möglichkeiten des CIS zusammen, die Ausführung der Builds zu organisieren.

- **Parallele Builds**
Auf dem CIS können mehrere Projekte parallel gebaut werden. Damit die Entwickler nicht zu lange auf die Ergebnisse des CIS warten müssen, sollte er dieses Kriterium erfüllen. Dieser Punkt wird umso wichtiger je mehr Entwickler in dem Unternehmen beschäftigt sind und damit öfter Builds ausgelöst werden. Dieser Punkt wurde am stärksten gewichtet.
- **Verteilte Builds**
Der CIS ist in der Lage Prozesse auf anderen Rechnern auszuführen. Dies geschieht meist durch Agenten, die auf den Rechnern gestartet und durch den CIS überwacht werden. Dadurch ist es z.B. möglich eine Anwendung unter anderen Systemumgebungen zu testen. Dieser Punkt erhielt deswegen die zweithöchste Gewichtung.
- **Auto-Update Agent**
Bei einem Update des CIS auf eine neuere Version, werden die Agenten ebenfalls aktualisiert. Dadurch vereinfacht sich die Instandhaltung der Agenten.
- **Abhängige Builds**
Der CIS ermöglicht es von einander abhängige Projekte zu verknüpfen, damit sie automatisch nacheinander ausgeführt werden. Dadurch eröffnen sich viele Möglichkeiten für die Umsetzung der Projekte im CIS. Beispielsweise könnte

¹³ http://en.wikipedia.org/wiki/Comparison_of_free_software_hosting_facilities

ein Prozess die Anwendung bauen und dann viele weitere Prozesse auslösen, welche im Anschluss die Anwendung testen oder veröffentlichen.

2.2.4 Build-Auslöser

Die Build-Prozesse können vom CIS durch unterschiedliche Ereignisse ausgelöst werden. Hierbei wurde der Auslöser „Versionsverwaltung“ am höchsten gewichtet, weil die Entwickler dadurch zeitnah über die Ergebnisse ihrer Änderungen unterrichtet werden können.

- **Zeit gesteuert**
Hierbei werden Builds periodisch in bestimmten Zeitabständen oder zu festen Zeitpunkten gestartet.
- **Manuell**
Manueller Start des Builds durch den Benutzer.
- **Versionsverwaltung**
Der CIS ist in der Lage die Builds bei Änderungen des überwachten Quellcodes in der Versionsverwaltung auszulösen.

2.2.5 Zugriffsschutz

Damit nicht jeder Benutzer unbeschränkten Zugriff auf die Benutzeroberfläche und Funktionen des CIS besitzt, wird eine Benutzerverwaltung benötigt. Da der CIS in einem geschützten Bereich des Firmennetzwerks arbeiten wird und die Anzahl der Entwickler (etwa 15) nicht sehr hoch ist, wurde den diese Kriterien insgesamt weniger hoch Gewichtet.

- **Benutzerverwaltung**
Durch eine Benutzerverwaltung lässt sich der Zugriff auf den CIS beschränken.
- **LDAP**
Der CIS unterstützt für die Benutzerverwaltung LDAP.

2.2.6 Veröffentlichung

Die Ergebnisse der CI-Prozesse sollen weitergegeben werden können. Dazu gehört die Unterrichtung der Entwickler bei Problemen und die Weitergabe der gebauten Anwendung.

- **E-Mail**
Benachrichtigung der Benutzer per E-Mail, wenn z.B. Builds fehlschlagen.
- **System Tray**
Rückmeldung über den Build-Fortschritt durch ein Programm in der Windows

System Tray.

- **FTP**
Der CIS bietet Funktionen, um Dateien auf einen FTP-Server hochzuladen.
- **SCP**
Der CIS bietet Funktionen, um Dateien auf einen SCP-Server hochzuladen.

2.2.7 Web-Oberfläche

Bietet der CIS eine web-basierte Oberfläche an, sind folgende Punkte für die Benutzerfreundlichkeit interessant. Dabei wurden insbesondere die ersten 4 Kriterien zur Verwaltung der Projekte höher gewichtet, weil in kleineren Unternehmen ein geringer Administrationsaufwand eine größere Rolle spielt. Aufgrund der relativ kleinen Anzahl an Mitarbeitern ist es wahrscheinlich nicht rentabel einzelne Personen ausschließlich für die Administration der Projekte einzusetzen.

- **Projekt hinzufügen**
Es lassen sich über die Oberfläche neue Projekte anlegen.
- **Projekt ändern**
Vorhandene Projekte lassen sich über die Oberfläche konfigurieren
- **Projekt kopieren**
Vorhandene Projekte lassen sich kopieren, um den Konfigurationsaufwand für neue Projekte zu reduzieren.
- **Verwaltung mehrerer Projekte**
Über die Oberfläche lassen sich mehrere Projekte verwalten.
- **Agenten verwalten**
Die Agenten für verteilte Builds lassen sich über die Oberfläche konfigurieren.
- **Build abbrechen**
Abbrechen eines Builds
- **Build pausieren**
Pausieren von Builds
- **Zugriff auf Build-Dateien**
Die Oberfläche erlaubt den Zugriff auf die Dateien des Builds, wie z.B. die erstellten Bibliotheken oder Programme.
- **Build Statistik**
Die Oberfläche bietet Statistiken zu den Builds an, die z.B. die Erfolgsrate aufzeigen.

2.2.8 GUI

Verwendet der CIS eine andere grafische Oberfläche, gelten die gleichen Kriterien wie für eine Web-Oberfläche. Wird das gleiche Kriterium in der Weboberfläche und GUI

erfüllt, erhält das CIS jedoch keine doppelte Punktzahl.

2.2.9 CLI

Die Build-Prozesse des CIS sollten sich per CLI abbrechen lassen. Dies könnte hilfreich sein, wenn aufgrund eines Programmfehlers die Benutzeroberfläche des CIS nicht mehr reagiert. Da dies im Idealfall nur selten auftreten sollte, wurde dieser Bereich insgesamt niedrig gewichtet.

- **Build abbrechen**
Abbruch eines Builds
- **Build pausieren**
Build pausieren

2.2.10 Darstellung von Test-Ergebnissen

Dieser Punkt behandelt die Möglichkeit des CIS die Ergebnisse von Testframeworks in der verwendeten Oberfläche darzustellen. Da die zentrale Sammlung und die Darstellung der Entwicklung der Ergebnisse zu den Hauptgründen zählt einen CIS einzusetzen, ist dieser Bereich relativ hoch gewichtet worden.

- **JUnit**
Testframework zur Erstellung von UnitTests für Java Programme.
- **NUnit**
Testframework zur Erstellung von UnitTests für .NET Programme.
- **PHPUnit**
Testframework zur Erstellung von UnitTests für PHP Programme.
- **Selenium**
Testframework zur Erstellung von Funktionstests für Webanwendungen
- **MS Test**
Testframework zur Erstellung von UnitTests für .NET Programme.

2.2.11 Direkter Build-Tool Support

Die meisten CIS können über die Kommandozeile andere Programme starten. Lassen sich die Build-Tools jedoch direkt über die Oberfläche aufrufen und konfigurieren, erleichtert dies die Einrichtung und Pflege der Projekte. Da der Aufruf der Build-Tools unter Umständen aufwendig werden kann, ist dieser Bereich insgesamt relativ hoch gewichtet worden.

- **Shell / cmd**

2.2.11 Direkter Build-Tool Support

Erlaubt Befehle über die Windows Kommandozeile cmd oder Linux shell abzuschicken.

- **Ant**
Erlaubt das Starten von Ant-Skripten für Java Projekte
- **Maven 1**
Buildmanagement Tool
- **Maven 2**
Buildmanagement Tool, Nachfolger von Maven 1
- **MS Build**
Build-Tool für .NET-Projekte
- **NAnt**
Portierung von Ant für die .NET Welt
- **devenv**
Kommandozeilen Programm von MS Visual Studio
- **Phing**
PHP Build-Tool

2.2.12 Direkter Test-Framework Support

Bei einer direkten Unterstützung von Test-Frameworks, können die Tests benutzerfreundlicher über die Oberfläche des CIS ausgeführt werden. Folgende Test-Frameworks sollten dabei von dem CIS unterstützt werden. Dieser Bereich ist jedoch relativ niedrig gewichtet, da sich alle Test-Frameworks relativ einfach über die Kommandozeile oder Build-Tools starten lassen.

- **Junit**
Unit Tests für Java Programme
- **Nunit**
Unit Tests für .NET Programme
- **PHPUnit**
Unit Tests für PHP Programme
- **Selenium**
Tests für Webanwendungen
- **MS Test**
Unit Tests für .NET Programme

2.3 Gewichtung

Die Kriterien sind nach Themenbereichen aufgeteilt. Jeder Bereich und jedes Kriterium werden nach ihrer Wichtigkeit für das Unternehmen (Schrittweite 0,01) gewichtet.

Erfüllt ein CIS beispielsweise in dem Bereich „Versionsverwaltung“ das Kriterium „Dateifilter“ erhält es dafür 0,15 Punkte.

$$\text{Gewichtung Bereich} * \text{Gewichtung Kriterium} * \text{Kriterium erfüllt} = \text{Punkte}$$

*Bsp.: 1 * 0,15 * 1 = 0,15*

Text 1: Evaluation - Bewertungs-Formel mit Beispiel

Eine Ausnahme bilden die Bereiche GUI und Web-Oberfläche. Werden von einem CIS in den Bereichen GUI und Web-Oberfläche die gleichen Kriterien erfüllt, werden die Punkte nicht addiert.

In der Spalte Einzelgewichtung sind die Gewichtungen der einzelnen Kriterien aufgeführt, die Summe der Einzelgewichtungen pro Bereich ist immer 1. Die Spalte Gesamtgewichtung enthält die Gewichtungen der Bereiche und die daraus resultierenden Gewichtungen der Kriterien. Durch die Summe der Bereiche ergibt sich eine Höchstpunktzahl von 9.

	Einzel- gewichtung	Gesamt- gewichtung
Code		0,8
Quellcode erhältlich	0,2	0,16
Redistribution erlaubt	0,2	0,16
Plugin-Support	0,6	0,48
Versionsverwaltung:		1
CVS	0,05	0,05
Subversion	0,65	0,65
Multiple Repositories pro Projekt	0,15	0,15
Dateifilter	0,15	0,15
Build-Ausführung:		0,8
Parallele Builds	0,6	0,48
Verteilte Builds	0,2	0,16
Auto-Update Agent Code	0,1	0,08
Abhängige Builds	0,1	0,08
Build-Auslöser:		1
Versionsverwaltung	0,4	0,4

2.3 Gewichtung

	Einzel- gewichtung	Gesamt- gewichtung
Zeitgesteuert	0,3	0,3
Manuell	0,3	0,3
Sicherheit:		0,5
Benutzerverwaltung	0,6	0,3
LDAP	0,4	0,2
Veröffentlichung:		1
Email	0,5	0,5
FTP	0,2	0,2
SCP	0,2	0,2
System Tray	0,1	0,1
CLI:		0,2
Build abbrechen	0,75	0,15
Build pausieren	0,25	0,05
GUI:		1,2
Projekt hinzufügen	0,15	0,18
Projekt ändern	0,1	0,12
Projekt kopieren	0,05	0,06
Verwaltung mehrerer Projekte	0,15	0,18
Agenten verwalten	0,05	0,06
Build abbrechen	0,3	0,36
Build pausieren	0,05	0,06
Zugriff auf Build-Dateien	0,05	0,06
Build Statistik	0,1	0,12
Web-Oberfläche:		1,2
Projekt hinzufügen	0,15	0,18
Projekt ändern	0,1	0,12
Projekt kopieren	0,05	0,06
Verwaltung mehrerer Projekte	0,15	0,18
Agenten verwalten	0,05	0,06
Build abbrechen	0,3	0,36
Build pausieren	0,05	0,06
Zugriff auf Build-Dateien	0,05	0,06

	Einzel- gewichtung	Gesamt- gewichtung
Build Statistik	0,1	0,12
Darstellung von Test-Ergebnissen:		1
JUnit	0,1	0,1
NUnit	0,35	0,35
PHPUnit	0,2	0,2
Selenium	0,2	0,2
MS Test	0,15	0,15
Direkter Build-Tool-Support:		1
Shell / cmd	0,4	0,4
Ant	0,1	0,1
Maven 1	0	0
Maven 2	0,05	0,05
MS Build	0,2	0,2
Nant	0,1	0,1
devenv (Visual Studio)	0,1	0,1
Phing	0,05	0,05
Direkter Test-Framework-Support:		0,5
JUnit	0,1	0,05
NUnit	0,35	0,18
PHPUnit	0,2	0,1
Selenium	0,2	0,1
MS Test	0,15	0,08

Tabelle 2.1: Evaluation - Gewichtung

2.4 Ergebnisse

Die folgende Tabelle zeigt mit Hand-Symbolen an, welche Kriterien von den untersuchten CIS erfüllt wurden. Zeigt der Daumen der Hand nach oben, wurde das Kriterium erfüllt, andernfalls nicht. Auf der linken Seite (in hellgrau) sind die kostenlosen OpenSource und auf der rechten Seite (in dunkelgrau) die kommerziellen CIS aufgelistet.

2.4 Ergebnisse

	CruiseControl	BuildBot	Anthill	Continuum	Hudson	CruiseControl .NET	CI Factory	Cruise	Anthill Pro	Bamboo	Pulse	TeamCity	BuildBeat CI	OpenMake Meister	OpenMake Mojo	FinalBuilder Server
<u>Code:</u>																
Quellcode erhältlich																
Redistribution erlaubt																
Plugin Support																
<u>Versionsverwaltung:</u>																
CVS																
Subversion																
Multiple Repositories pro Projekt																
Dateifilter																
<u>Build-Ausführung:</u>																
Parallele Builds																
Verteilte Builds																
Auto-Update Agent Code																
Abhängige Builds																
<u>Build-Auslöser:</u>																
Versionsverwaltung																
Zeitgesteuert																
Manuell																
<u>Sicherheit:</u>																

	CruiseControl	BuildBot	Anthill	Continuum	Hudson	CruiseControl .NET	CI Factory	Cruise	Anthill Pro	Bamboo	Pulse	TeamCity	BuildBeat CI	OpenMake Meister	OpenMake Mojo	FinalBuilder Server
Benutzerverwaltung																
LDAP																
<u>Veröffentlichung:</u>																
Email																
FTP																
SCP																
System Tray																
<u>CLI:</u>																
Build kill																
Build pause																
<u>GUI:</u>																
Projekt hinzufügen																
Projekt ändern																
Projekt kopieren																
Verwaltung mehrerer Projekte																
Agenten verwalten																
Build kill																
Build pause																
Zugriff auf Build-Dateien																
Build Statistik																

2.4 Ergebnisse

	CruiseControl	BuildBot	Anthill	Continuum	Hudson	CruiseControl .NET	CI Factory	Cruise	Anthill Pro	Bamboo	Pulse	TeamCity	BuildBeat CI	OpenMake Meister	OpenMake Mojo	FinalBuilder Server
Web-Oberfläche:																
Projekt hinzufügen																
Projekt ändern																
Projekt kopieren																
Verwaltung mehrerer Projekte																
Agenten verwalten																
Build kill																
Build pause																
Zugriff auf Build-Dateien																
Build Statistik																
Darstellung von Test-Ergebnissen:																
JUnit																
NUnit																
PHPUnit																
Selenium																
MS Test																
Direkter Build-Tool Support:																
Shell / cmd																
Ant																

	CruiseControl	BuildBot	Anthill	Continuum	Hudson	CruiseControl .NET	CI Factory	Cruise	Anthill Pro	Bamboo	Pulse	TeamCity	BuildBeat CI	OpenMake Meister	OpenMake Mojo	FinalBuilder Server
Maven 1																
Maven 2																
MS Build																
Nant																
devenv (Visual Studio)																
Phing																
<u>Direkter Test-Framework Support:</u>																
JUnit																
NUnit																
PHPUnit																
Selenium																
MS Test																

Tabelle 2.2: Evaluation - Gesamtübersicht

2.4 Ergebnisse

Die folgende Tabelle zeigt die Gesamtpunktzahl der einzelnen CIS. Die kostenlosen wurden hellgrau und die kommerziellen dunkelgrau unterlegt:

Platz	Punkte	CIS	Platz	Punkte	CIS
1	7,89	Hudson	9	6,63	Pulse
2	7,71	Anthill Pro	10	6,33	FinalBuilder Server
3	7,68	Bamboo	11	5,92	CI Factory
4	7,45	OpenMake Meister	12	5,85	BuildBeat CI
5	7,12	OpenMake Mojo	13	5,35	Continuum
6	6,68	Cruise	14	4,97	CruiseControl .NET
7	6,68	TeamCity	15	4,56	BuildBot
8	6,63	CruiseControl	16	2,68	Anthill

Tabelle 2.3: Evaluation - Rangliste mit Gesamtpunktzahl (Durchschnitt 6,26 Punkte)

Die folgende Tabelle zeigt die detaillierte Punktevergabe für die besten 3 CIS. Die kostenlosen wurden hellgrau und die kommerziellen dunkelgrau unterlegt:

	Hudson	Anthill Pro	Bamboo
Code:			
Quellcode erhältlich	0,16	0	0,16
Redistribution erlaubt	0,16	0	0
Plugin Support	0,48	0,48	0,48
Versionsverwaltung:			
CVS	0,05	0,05	0,05
Subversion	0,65	0,65	0,65
Multiple Repositories pro Projekt	0,15	0,15	0,15
Dateifilter	0,15	0,15	0,15
Build-Ausführung:			
Parallele Builds	0,48	0,48	0,48

	Hudson	Anthill Pro	Bamboo
Verteilte Builds	0,16	0,16	0,16
Auto-Update Agent Code	0,08	0,08	0,08
Abhängige Builds	0,08	0,08	0,08
Build-Auslöser:			
Versionsverwaltung	0,4	0,4	0,4
Zeitgesteuert	0,3	0,3	0,3
manuell	0,3	0,3	0,3
Sicherheit:			
Benutzerverwaltung	0,3	0,3	0,3
LDAP	0,2	0,2	0,2
Veröffentlichung:			
Email	0,5	0,5	0,5
FTP	0,2	0	0,2
SCP	0,2	0,2	0,2
System Tray	0,1	0,1	0,1
CLI:			
Build kill	0	0	0
Build pause	0	0	0
GUI:			
Projekt hinzufügen	0	0	0
Projekt ändern	0	0	0
Projekt kopieren	0	0	0
Verwaltung mehrerer Projekte	0	0	0
Agenten verwalten	0	0	0
Build kill	0	0	0
Build pause	0	0	0
Zugriff auf Build-Dateien	0	0	0
Build Statistik	0	0	0
Web-Oberfläche:			
Projekt hinzufügen	0,18	0,18	0,18

2.4 Ergebnisse

	Hudson	Anthill Pro	Bamboo
Projekt ändern	0,12	0,12	0,12
Projekt kopieren	0,06	0,06	0,06
Verwaltung mehrerer Projekte	0,18	0,18	0,18
Agenten verwalten	0,06	0,06	0,06
Build kill	0,36	0,36	0,36
Build pause	0	0,06	0
Zugriff auf Build-Dateien	0,06	0,06	0,06
Build Statistik	0,12	0,12	0,12
Darstellung von Test-Ergebnissen:			
JUnit	0,1	0,1	0,1
NUnit	0,35	0,35	0,35
PHPUnit	0	0	0,2
Selenium	0,2	0,2	0
MS Test	0	0	0
Direkter Build-Tool Support:			
Shell / cmd	0,4	0,4	0,4
Ant	0,1	0,1	0,1
Maven 1	0	0	0
Maven 2	0,05	0,05	0,05
MS Build	0,2	0,2	0,2
Nant	0,1	0,1	0,1
devenv (Visual Studio)	0	0,1	0,1
Phing	0,05	0	0
Direkter Test-Framework Support:			
JUnit	0	0,05	0
NUnit	0	0,18	0
PHPUnit	0	0	0
Selenium	0,1	0,1	0
MS Test	0	0	0
Summe:	7,89	7,71	7,68

Tabelle 2.4: Evaluation - Detaillierte Punktetabelle der Top 3

2.5 Lizenzkosten

Ein vor allem für kleinere Unternehmen wichtiger Punkt sind die Lizenzkosten der CIS. Hierbei gibt es bei den kommerziellen Herstellern sehr unterschiedliche Konzepte. Meistens handelt es sich um einmalige Kosten, die sich nach der Anzahl der CIS Installationen und Benutzer staffeln. Werden verteilte Builds unterstützt, spielt die Menge der unterstützten Agenten ebenfalls eine Rolle. Andere Modelle sehen dafür jährliche Gebühren vor.

Auf Platz 2 der Bewertungs-Rangliste hatte es Anthill Pro von Urbancode geschafft. Leider waren im Zeitraum der Evaluation auf der Internetseite des Herstellers keine Angaben zu der Höhe der Lizenzkosten zu finden und auf E-Mail-Anfragen bzgl. der Kosten wurde nicht reagiert-. Daher wurde Anthill Pro für die weitere Evaluation nicht mehr in Betracht gezogen.

Da zur Zeit der Evaluierung in der Firma eWorks GmbH etwa 15 Mitarbeiter angestellt waren, wurden die Lizenzkosten für 10-20 Benutzer betrachtet. Weiterhin war es zu der Zeit noch unklar, ob für die Einrichtung der Projekte Agenten für verteilte Builds benötigt werden, daher wurden die Kosten einmal ohne und einmal mit 5 Agenten verglichen.

ThoughtWorks verlangt für sein CIS Cruise eine jährliche Gebühr, welches in der Grundausstattung bereits 10 Agenten enthält. Bamboo besitzt genauso wie Pulse mit der günstigsten Lizenz bereits einen Agenten. TeamCity bringt in der günstigsten Version bereits 3 Agenten mit.

Die folgende Tabelle listet die Lizenzkosten der kommerziellen CIS auf:

	Cruise	Bamboo	Pulse	TeamCity	BuildBeat CI	OpenMake Meister	OpenMake Mojo	FinalBuilder Server
10 Benutzer	2.221 €	1.527 €	1.041 €	1.180 €	1.728 €	13.151 €	6.905 €	590 €
15 Benutzer	3.227 €	1.527 €	1.041 €	1.180 €	1.728 €	19.727 €	10.358 €	1.388 €
20 Benutzer	4.233 €	1.527 €	1.041 €	1.180 €	1.728 €	26.303 €	13.811 €	1.388 €
10 Benutzer + 5 Agenten	2.221 €	2.776 €	1.735 €	1.534 €	5.719 €	19.727 €	10.358 €	590 €
15 Benutzer + 5 Agenten	3.227 €	2.776 €	1.735 €	1.534 €	5.719 €	26.303 €	13.811 €	1.388 €
20 Benutzer + 5 Agenten	4.233 €	2.776 €	1.735 €	1.534 €	5.719 €	32.878 €	17.263 €	1.388 €

Tabelle 2.5: Evaluation Lizenzkosten

2.6 Direktvergleich der Favoriten

Da Anthill Pro aufgrund der unklaren Lizenzkosten nicht mehr in Frage kommt, wurden für die finale Entscheidung die zwei CIS Hudson und Bamboo erneut im direkten Vergleich getestet. Dabei wurde das Augenmerk insbesondere auf die Benutzerfreundlichkeit und Darstellung der Oberfläche gerichtet. Um diese beiden Aspekte gut vergleichen zu können, wurde in beiden CIS das selbe Projekt eingerichtet. Es handelte sich dabei um ein Java-Projekt mit einem Ant Build-Script und JUnit Tests. Damit war es beispielsweise möglich die Darstellung und Handhabung von Build- und Test-Ergebnissen zu beurteilen.

Beide CIS wurden für den Vergleich auf einer virtuellen Maschine des Herstellers VMWare¹⁴ installiert. Zum Einsatz kam dabei VMWare Workstation der Version 6.5.3 und als Betriebssystem für die Maschine diente Windows Server 2008 von Microsoft¹⁵.

2.6.1 Bamboo

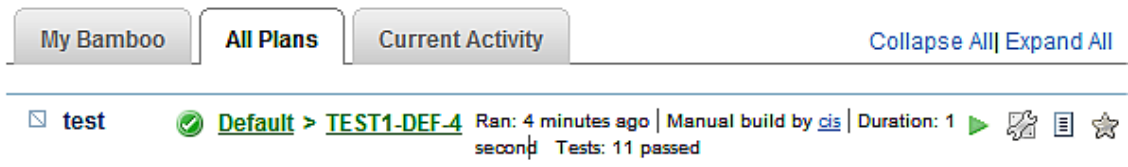
Bei Bamboo ist die Weboberfläche besser strukturiert. Die Informationen werden

¹⁴ <http://www.vmware.de>

¹⁵ <http://www.microsoft.de>

ansprechend präsentiert und die Oberfläche bietet mehrere Funktionen, die bei Hudson fehlen.

Das folgende Abbild zeigt die Gesamtübersicht von Bamboo:



Successful builds are green, failed builds are red.

Abbildung 1: Bamboo - Gesamtübersicht

Im Vergleich dazu die Gesamtübersicht von Hudson:



Abbildung 2: Hudson - Gesamtübersicht

Das folgende Abbild zeigt die Projekt-Zusammenfassung von Bamboo:

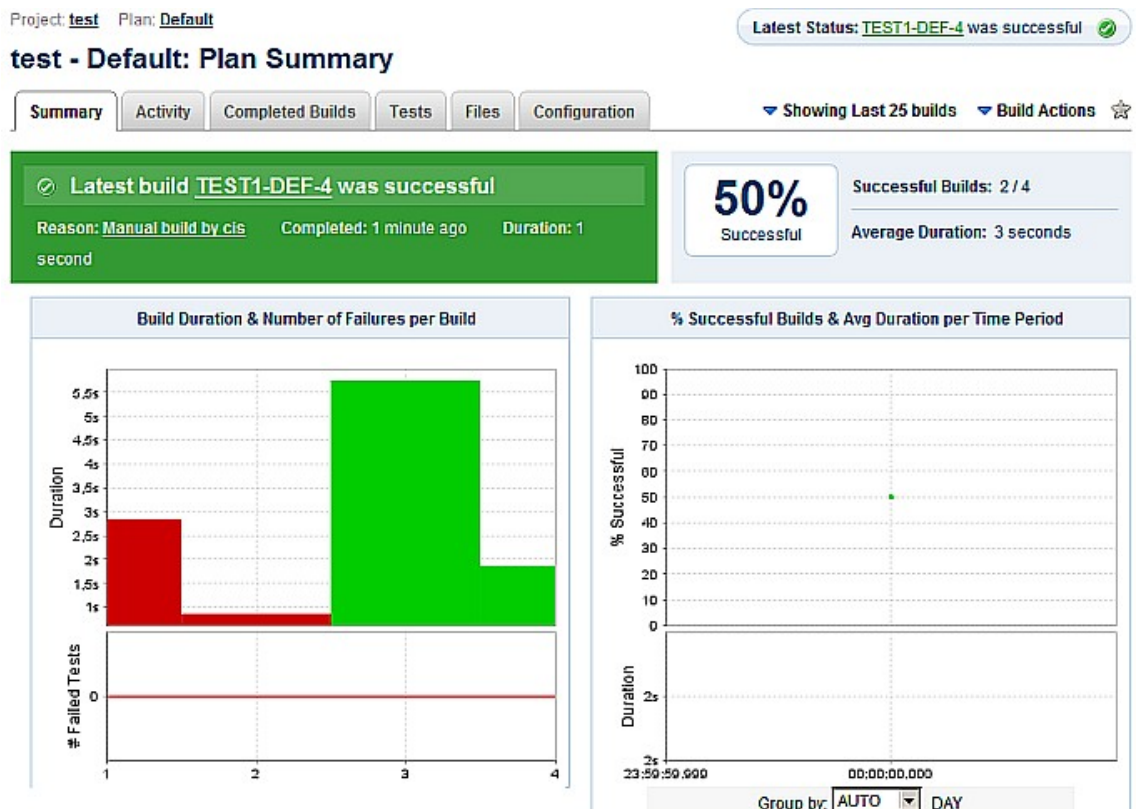


Abbildung 3: Bamboo - Projekt Zusammenfassung

Im Vergleich dazu die Projekt-Zusammenfassung von Hudson:

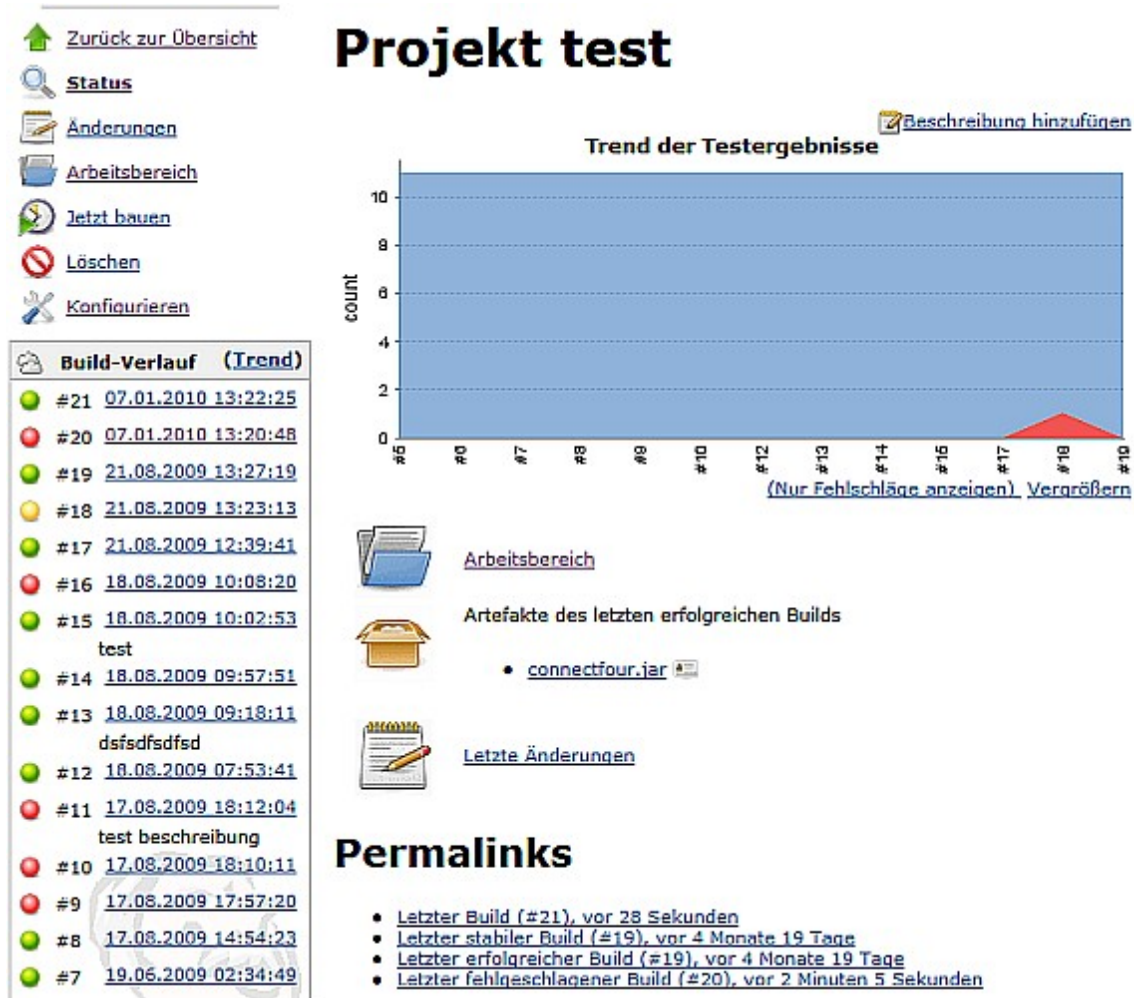


Abbildung 4: Hudson - Projekt-Zusammenfassung

Bamboo kann Berichte erstellen, in denen gehören Reportfunktionen und Statistiken zu den Builds/Tests der Projekte und Aktivitäten der Benutzer. Hudson bietet ebenfalls Diagramme auf Ebene der Projekte an, jedoch sind diese nicht flexibel über Zeitraum und Auswahl mehrerer Projekte einstellbar (Abb. 5).

Über die Historie einzelner Tests lassen sich Diagramme erstellen, diese zeigen die Anzahl der Fehlschläge und die Testdauer an. Zusätzlich bietet Bamboo auch eine Übersicht über die 10 Tests mit den meisten Fehlschlägen oder der längsten Laufdauer an.

Schlagen Builds oder Tests fehl, lässt sich in der Übersicht zu dem Projekt feststellen mit welcher Build-Version das Problem auftrat und wieder behoben wurde. Hudson liefert die Information über den ersten Fehlschlag von Tests, nicht jedoch zu den Builds.

Um bestimmte Builds einfacher finden zu können, lassen sich diese mit Hilfe von Labels kategorisieren. Dazu wird dem Build manuell oder automatisch ein Link zugewiesen, über den alle Builds der gleichen Kategorie abrufbar sind.

Zu den Builds lassen sich von den Benutzern der Oberfläche Kommentare hinzufügen. Hudson erlaubt hier nur das Hinzufügen einer Beschreibung, welche man aber auch für Kommentare nutzen könnte.

Die Verwaltung der Benutzer-Rechte und Agenten ist bei Bamboo übersichtlicher gestaltet. Die Benutzerrechte werden in einer Gesamtübersicht über alle Projekte und Benutzer vergeben. Projektspezifische Einstellungen sind nicht wie bei Hudson auf die Konfigurationen der einzelnen Projekte verteilt. Lokale Build-Prozesse werden hierbei ebenfalls als Agenten aufgefasst und können zur Verwendung unterschiedlicher Builder-Versionen und Tools konfiguriert werden. Beim Erstellen von Projekten werden diese Informationen genutzt. So wird geprüft, ob es für die Projektbedingungen bereits Agenten gibt, die diese erfüllen.

Ein Punkt der an Bamboo stört, ist die Abhängigkeit der Projekt-Pläne von Buildscripten. Von Haus aus gibt es über die Oberfläche keine Möglichkeit neben dem Aufruf eines Build-Tools noch weitere Aktionen durchzuführen. Dies muss über das Buildscript geschehen. Es gibt allerdings ein Plugin, womit man optional vor und nach dem Build einen Kommandozeilenbefehl ausführen kann. Diese Limitierung erschwert jedoch die Konfiguration von Projekten und führt dazu, dass man sich unter Umständen bei komplizierteren Vorgängen wie Veröffentlichung der Anwendung oder Test-Installation notgedrungen mit mehreren Plänen behelfen muss, statt einfach in einem Plan weitere Aktionen durchzuführen.

Ein wenig unpraktisch ist auch die Tatsache, dass zu jedem Plan immer eine Versionsverwaltung angegeben werden muss. Man kann also z.B. nicht einen Plan erstellen, der einfach nur ein Programm aufruft ohne vorher die überwachten Dateien auf Änderungen zu überprüfen. In dem Fall müsste man für die gewünschten Aktionen ein Skript schreiben und in die Versionsverwaltung einpflegen.

2.6.2 Hudson

Für Hudson spricht die einfache Konfiguration des Buildprozesses, hier lassen sich eine

2.6.2 Hudson

beliebige Anzahl von Builder-Aufrufen und Kommandos aneinander reihen, ohne dafür zwangsweise ein Buildscript schreiben zu müssen. Das beschleunigt die Erstellung von Projekten und ermutigt die Entwickler neue Projekte anzulegen.

Das folgende Bild zeigt das Anlegen der Build-Schritte:

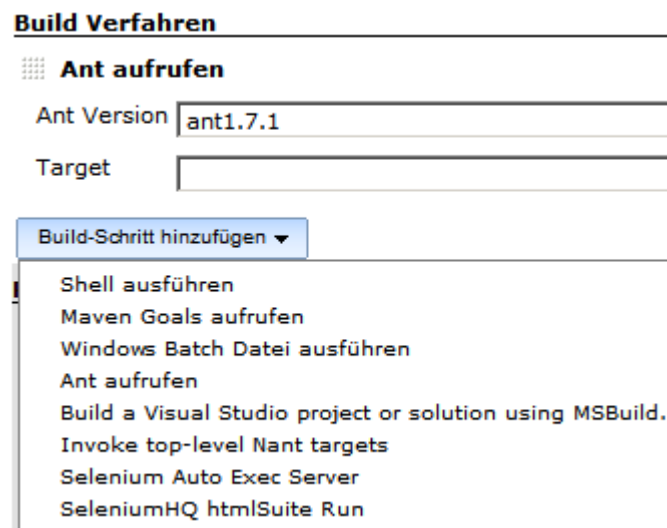


Abbildung 6: Hudson - Konfiguration der Build-Schritte

Die Konfiguration wird zusätzlich durch die integrierte Hilfe zu den einzelnen Einstellungen vereinfacht. Klickt man auf die blauen Fragezeichen neben den Feldern, wird ein Text mit einer Erklärung eingeblendet. Dies zeigt der folgende Bildausschnitt:

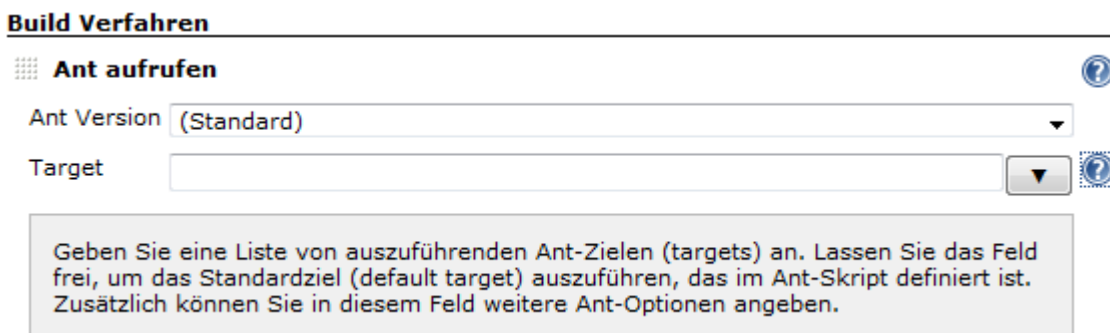


Abbildung 7: Hudson - Hilfe in der Konfiguration

Durch die flexible Art der Projekte, könnten man diese auch zum Überwachen anderer Prozesse einsetzen, die keine Software bauen. So könnte man z.B. über einen Agenten auf einem anderen Rechner einen regelmäßigen Backup-Prozess in Gang setzen oder in Abhängigkeit bestimmter Ereignisse andere Jobs anstoßen.

Für archivierte Build-Artefakte bietet Hudson ein Fingerprint-Feature an, damit lässt sich bei Problemen genau feststellen aus welchem Build ein Artefakt entstanden ist. Dies

ist bei Abhängigkeiten zwischen verschiedenen Projekten interessant, bei denen z.B. Bibliotheken untereinander ausgetauscht werden.

Zu jedem Projekt werden feste Links zu bestimmten Builds angeboten, die sich nutzen lassen, um z.B. auf einer Homepage stets den Link zu den aktuellsten Build-Dateien anzubieten. So gibt es feste Links für den letzten stabilen, erfolgreichen, fehlgeschlagenen oder zuletzt durchgeführten Build.

Ein weiterer Vorteil von Hudson ist die große Anzahl an Plugins, die durch die aktive Community erstellt wurden. So ist es z.B. möglich Selenium-Tests zu integrieren und ein Selenium Grid zu verwalten oder Links zu Mantis Bugtracker Meldungen zu integrieren und Kommentare zu den Builds in Bugtracker eintragen zu lassen. Über ein VMWare Plugin lassen sich Pre- und Post-Build Aktionen festlegen, wodurch sich z.B. VMWare Server Instanzen starten und Snapshots laden lassen. Weiterhin gibt es zahlreiche Erweiterungen zu Metrik-Tools, um die Ergebnisse in Statistiken und Diagrammen darzustellen.

Was etwas an Hudson stört, ist die Navigation in der Weboberfläche, die teilweise etwas mehr Klicks erfordert um an die gewünschte Informationen zu kommen. Alle wichtigen Informationen zu den Builds und Tests sind vorhanden, könnten aber etwas übersichtlicher dargestellt werden. Aber auch zu diesem Problem gibt es bereits ein Plugin, womit sich mehrere Ansichten mit Informationen wie die Projektübersicht und Diagramme individuell einstellen lassen.

Um die Unterstützung von Plugin-Erweiterungen noch weiter zu unterstreichen, gibt es in Hudsons Weboberfläche eine Plugin-Verwaltung. Hier kann man auf der Hudson Homepage verfügbare Plugins herunterladen und installierte Plugins aktualisieren oder deaktivieren.

2.6.3 Zusammenfassung

Zusammengefasst bietet Hudson mehr Funktionen bei der Konfiguration der Projekte und durch die zahlreich vorhandenen Plugins werden viele Funktionen erweitert und weitere Tools unterstützt. Bamboo hingegen hat eine schöner gestaltete Oberfläche mit zusätzlichen Funktionen für den Umgang mit den Informationen, wie einstellbare Statistiken/Diagramme über Projekte/Tests/Entwickler und das Hinzufügen von Labels

2.6.3 Zusammenfassung

und Kommentaren zu den Build-Ergebnissen.

In den Erweiterungsmöglichkeiten sind Bamboo und Hudson ebenbürtig. Beide CIS bieten über so genannte Extensionpoints viele Möglichkeiten den Funktionsumfang durch Plugins zu erweitern.

Für beide CIS gibt es auch bereits zahlreiche Plugins, die negative Punkte in der Grundausstattung abschwächen.

Bezieht man bereits vorhandene Plugins in den Vergleich mit ein, sehen die Hauptunterschiede etwa so aus:

Bamboo:

- bessere Struktur und Gestaltung der Oberfläche
- Verwaltung der Benutzerrechte einfacher
- Report Funktionen zu einzelnen Tests und Benutzern
- Kommentarfunktion für Buildergebnisse

Hudson:

- einfache, flexible und umfangreiche Projekt Konfiguration
- Fingerprinting für Build-Artefakte
- feste Links zum letzten erfolgreichen Build und dessen Artefakte
- zusätzliche Plugins für:
 - Selenium¹⁶-Tests
 - Verwaltung eines Selenium-Grid
 - Unterstützung von mehr Metrik-Tools
 - Mantis¹⁷ Bugtracker
 - VMWare Server

Die Unterstützung für die Anwendungen Bugtracker, VMWare und das Selenium Testframework sind besonders interessant, da diese in der Firma eWorks GmbH teilweise sehr häufig eingesetzt werden.

Neben den Erweiterungsmöglichkeiten durch Plugins war die einfache Konfiguration der Projekte eines der Hauptkriterien für die Auswahl des CIS. Diesen Punkt erfüllt Hudson weitaus zufriedenstellender als Bamboo. Durch die einfache und flexible Konfiguration werden Entwickler ermutigt spontan eigene Projekte zu erstellen und so

¹⁶ [Http://www.selenium.de](http://www.selenium.de)

¹⁷ [Http://www.mantis.de](http://www.mantis.de)

z.B. ihre Software unter anderen Bedingungen zu testen.

Eines der Hauptgründe für die Verwendung eines CIS, war die Überwachung der Softwarequalität der einzelnen Projekte. Damit sollen Fragen beantwortet werden, wie: Laufen alle Tests erfolgreich durch? Wie hoch ist die Codeabdeckung der Test? Wie hoch ist die Codekomplexität, Kopplung zwischen den Klassen oder werden Coderichtlinien eingehalten?

Die erste Frage können beide CIS beantworten, wobei Bamboo durch die Report-Funktion auf Test-Ebene einen leichten Vorteil hat. Alles andere ist nicht Teil beider CIS, sondern wird über externe Tools getestet. Die dabei entstehenden Berichte können dann mit Hilfe von Plugins durch die CIS ausgelesen und dargestellt werden. An dieser Stelle liegt der Vorteil auf Hudsons Seite, da hier das Angebot an unterstützten Metrik-Tools größer ist.

Alle wichtigen Informationen zu den Builds und Tests sind in Hudsons Weboberfläche enthalten und fehlende Features, wie Diagramme über einzelne Tests oder Benutzer könnte man evtl. über ein Plugin ergänzen. Durch die bereits vorhandenen Plugins für viele in der Firma eWorks benutzte Tools entsteht ein größerer Nutzen, da man sich auf die Entwicklung weiterer Erweiterungen konzentrieren oder diese noch weiter ausbauen kann.

Aufgrund der höheren Flexibilität und Benutzerfreundlichkeit bei der Konfiguration der Projekte und der zahlreichen Plugins, die den Funktionsumfang von Hudson weiter erhöhen, fiel die Entscheidung zu Gunsten von Hudson aus.

2.7 Eigenentwicklung

Der Funktionsumfang ist heutzutage selbst bei kostenlosen CI-Servern so hoch, dass eine Eigenentwicklung nicht mehr notwendig ist. Die meisten Vorgänge eines CI-Prozesses werden entweder direkt durch den Server unterstützt oder lassen sich mit dessen Hilfe durch weitere Anwendungen in Gang setzen. Werden für ein Projekt einmal Sonderlösungen notwendig, stehen oft umfangreiche Möglichkeiten zur Verfügung, die Funktionen des CI-Servers durch Plugins zu erweitern. Es ist also sinnvoller einen CIS nach den Bedürfnissen des Unternehmens auszuwählen und die gesparte Zeit für eine Eigenentwicklung in die Erweiterungen zu investieren.

3 Erweiterung des Continuous Integration Servers

Eines der Hauptargumente für die Entscheidung Hudson als CIS einzusetzen war das große Angebot an Plugins, die Hudson um weitere Funktionen erweitern. In den folgenden Unterkapiteln werden die interessantesten Plugins unterteilt nach Kategorien vorgestellt. Zu jedem Plugin ist jeweils angegeben, ob es bei der Umsetzung der Projekte im Kapitel 4.2 eingesetzt wurde.

Durch die aktive Community sind seit der Veröffentlichung von Hudson im Jahre 2005 über 200 Plugins¹⁸ entstanden. Um die Community zu unterstützen und die Verbreitung der Plugins zu fördern, stehen den Entwicklern von Plugins verschiedene Dienste¹⁹ zur Verfügung. Die Bereitstellung der Dienste ist an keinerlei Bedingungen geknüpft, man muss nur eine E-Mail an *dev@hudson.dev.java.net* schicken und den Benutzernamen seines java.net-Zugangs²⁰ nennen. Ein solcher Zugang kann von jeder Person kostenlos angelegt werden.

Die Plugin-Entwickler haben die Möglichkeit den offiziellen Hudson Subversion-Server²¹ für die Versionsverwaltung ihres Quellcodes zu nutzen und damit für andere Entwickler zugänglich zu machen. Mit einem Subversion-Client kann sich jeder Interessent den Quellcode der Plugins unter der Adresse *https://hudson.dev.java.net/svn/hudson/trunk/hudson/plugins/<Plugin-ID>* und mit dem Benutzernamen „guest“ herunterladen.

Die einzelnen Plugin-Releases werden in Form von HPI-Dateien ebenfalls über den offiziellen Server (unter *https://hudson.dev.java.net/servlets/ProjectDocumentList*) bereitgestellt.

Die Entwickler können außerdem zur Dokumentation ihrer Plugins auf der Wiki-Homepage von Hudson eigene Seiten erstellen. Auf diesen Seiten wird üblicherweise der Verwendungszweck und die Handhabung des Plugins erläutert und enthält oft auch eine Auflistung der Änderungen zu jeder Release-Version.

18 <http://wiki.hudson-ci.org/display/HUDSON/Plugins>

19 <http://wiki.hudson-ci.org/display/HUDSON/Hosting+Plugins>

20 <https://www.dev.java.net/servlets/Join>

21 Subversion-Server von Hudson: <https://hudson.dev.java.net/svn>

3.1 Plugin-Kategorien

Ist man auf der Suche nach interessanten Plugins, beginnt man am besten im Plugin-Bereich²² von Hudsons Wiki-Seiten. Dort befindet sich eine nach Kategorien aufgeteilte Liste von veröffentlichten Plugins mit einer kurzen Beschreibung und einem Verweis auf dessen Dokumentation. Die Kategorien unterscheiden nach den Funktionen von Hudson, die von den Plugins erweitert werden.

3.1.1 Source code management

Hudson unterstützt in der Grundausstattung bereits die Versionsverwaltungssysteme Subversion und CVS.

Plugins dieser Kategorie erweitern die Unterstützung um weitere Systeme zur Versionsverwaltung. Dazu gehören inzwischen z.B. Git²³, Team Foundation Server²⁴, Bazaar²⁵, Mercurial²⁶, Perforce²⁷ und Visual SourceSafe²⁸.

Des Weiteren gibt es Plugins, die eine Versionsverwaltung nur simulieren. Ein Beispiel dafür ist das Plugin „File System SCM“, welches auf Änderungen von Dateien im Dateisystem reagiert und einen Build-Vorgang auslösen kann.

3.1.2 Build-Triggers

Zu dieser Kategorie gehören alle Plugins, die das Auslösen von Builds beeinflussen können.

Ein interessantes Beispiel ist das Plugin „Join“. In Hudson kann man einen Job so konfigurieren, dass er weitere Jobs auslöst. Das Zusammenführen mehrerer Jobs ist jedoch nicht möglich. Dieses Plugin schließt diese Lücke und löst einen Job erst dann aus, wenn vorher eine Reihe definierter Jobs erfolgreich beendet wurden.

Mit dem Plugin „URL Change Trigger“ lässt sich ein Job auslösen, wenn sich der Inhalt einer definierten URL geändert.

Die Plugins „Naginator“ und „Retry Failed Builds“ wiederholen fehlgeschlagene Jobs

22 Plugin-Dokumentationen: <http://wiki.hudson-ci.org/display/HUDSON/Plugins>

23 Git: <http://git.or.cz/>

24 TFS: [http://msdn.microsoft.com/en-us/library/aa730884\(VS.80\).aspx](http://msdn.microsoft.com/en-us/library/aa730884(VS.80).aspx)

25 Bazaar: <http://bazaar-vcs.org/>

26 Mercurial: <http://selenic.com/mercurial/>

27 Perforce: <http://perforce.com/>

28 Visual SourceSafe: <http://msdn.microsoft.com/de-de/library/3h0544kx%28VS.80%29.aspx>

automatisch in einer festgelegten Zeitperiode.

Mit dem Plugin „Parameterized Trigger“ kann ein Job andere Jobs auslösen und dabei zusätzlich Parameter-Werte übergeben. Bei der normalen Konfiguration von Job-Abhängigkeiten ist dies nicht möglich.

3.1.3 Build-Tools

Zur Ausführung des eigentlichen Build-Prozesses in den Jobs, unterstützt Hudson bereits die Build-Tools Maven und Ant. Des Weiteren lassen sich Shell-Skripte und Windows Batch-Kommandos ausführen.

Plugins dieser Kategorie erweitern die Unterstützung um weitere Build-Tools und fügen weitere Build-Funktionen hinzu, die bei der Konfiguration der Jobs verwendet werden können. Mittlerweile werden viele weitere Build-Tools und -Skripte unterstützt, dazu gehören z.B. MSBuild, Nant, Phing, Groovy, PowerShell, Python, Rake und Ruby.

Bei der Umsetzung der Projekte wurden das Plugin „MSBuild“ eingesetzt, welche die Verwendung des gleichnamigen Build-Tools unterstützt. Mit MSBuild lassen sich unter anderem die Projekt-Dateien der Entwicklungsumgebung MS Visual Studio zum Bauen der Anwendung verwenden.

Durch das Plugin „Seleniumhq²⁹“ lassen sich in der Job-Konfiguration Build-Schritte hinzufügen, die Tests des Testframeworks Selenium ausführen. Das Plugin erweitert außerdem die Weboberfläche. Der Projekt-Übersicht wird ein Graph über die Testergebnisse und einen Verweis zum letzten Test-Bericht hinzugefügt. Der Graph wird in der folgenden Abbildung auf der rechten Seite dargestellt:

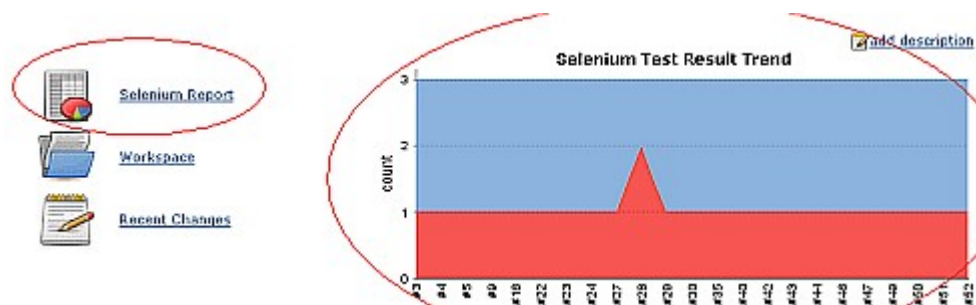


Abbildung 8: Plugin Seleniumhq - Projekt

Die Build-Übersicht wird um eine kurze Zusammenfassung der Testergebnisse

²⁹ <http://wiki.hudson-ci.org/display/HUDSON/Seleniumhq+Plugin>

3.1.3 Build-Tools

erweitert und verfügt über einen Verweis zum genauen Testbericht. Dies ist in der folgenden Abbildung ersichtlich:

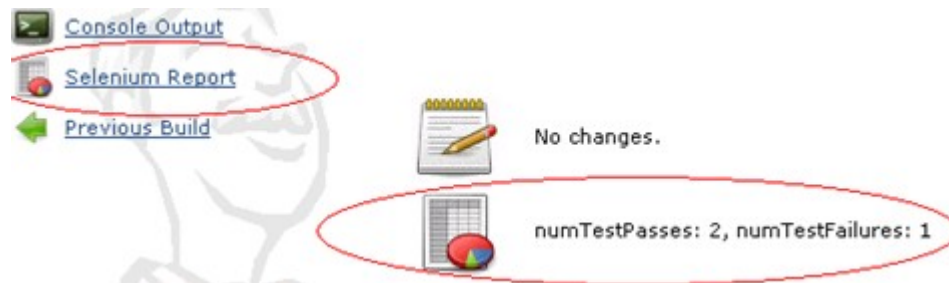


Abbildung 9: Plugin Seleniumhq - Build

3.1.4 Build-Wrappers

Mit den Plugins dieser Kategorie werden Funktionen hinzugefügt, die vor und/oder nach dem Build-Vorgang der Jobs ausgeführt werden können.

Bei der Umsetzung der Projekte (Kapitel 4.2) wurde unter anderem das Plugin „Build-timeout³⁰“ eingesetzt. Damit kann man in der Job-Konfiguration eine Zeit festlegen, nach der ein Job automatisch abgebrochen werden soll. Dies ist besonders hilfreich, da die Anzahl paralleler Jobs in Hudson begrenzt ist und ein „hängender“ Build-Prozess die Ausführung anderer Jobs behindern kann.

Mit dem Plugin „Setenv³¹“ lassen sich in einem Job Umgebungsvariablen festlegen, die in den Build-Schritten der Konfiguration referenziert werden können. Leider kann man die Variablen nicht in den Pre- und Post-Build Aktionen (z.B. bei der Archivierung der Artefakte) benutzen, deshalb wurde es bei der Umsetzung der Projekte nicht eingesetzt.

Das Plugin „VMWare³²“ erlaubt es vor dem Start des Build-Vorganges eine virtuelle Maschine der Anwendung VMWare zu starten und nach dem Build wieder zu stoppen. Dieses Plugin kann besonders für Jobs hilfreich sein, in denen z.B. eine Anwendung unter verschiedenen Systemumgebungen getestet werden soll. Die virtuelle Maschine könnte nur bei Bedarf gestartet werden und verbraucht somit auf der physischen Maschine keine Ressourcen. Ein möglicher Nachteil ist die längere Dauer des ganzen Build-Vorganges, falls die virtuelle Maschine erst gestartet werden muss. In den umgesetzten Projekten wurde das Plugin nicht eingesetzt, da die virtuellen Test-Server nicht heruntergefahren werden sollten.

30 <http://wiki.hudson-ci.org/display/HUDSON/Build-timeout+Plugin>

31 <http://wiki.hudson-ci.org/display/HUDSON/Setenv+Plugin>

32 <http://wiki.hudson-ci.org/display/HUDSON/VMware+plugin>

Das Plugin „Release“³³ fügt einem Job weitere Funktionen hinzu, die bei einem manuellen Release-Vorgang hilfreich sein können. So können weitere Parameter definiert und zusätzliche Build-Schritte vor und nach dem eigentlichen Build-Vorgang der Anwendung ausgeführt werden.

Ein Release erhält eine frei definierbare Versionsnummer und wird in der Build-Übersicht der Weboberfläche gekennzeichnet. Zusätzlich wird der Schreibschutz des Builds aktiviert, damit dieser z.B. nicht durch automatische Vorgänge (wie das Verwerfen alter Builds) von Hudson gelöscht wird.



Abbildung 10: Plugin Release - Build Verlauf

Durch das Plugin „Dashboard View“ erhält man zusätzlich eine Übersicht über die letzten Releases.

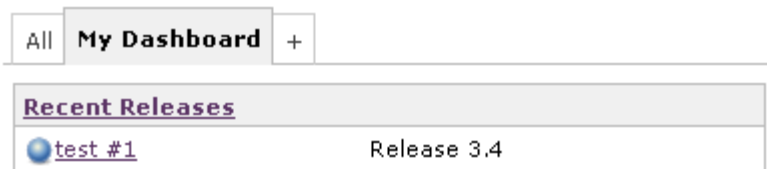


Abbildung 11: Plugin Release - Dashboard Übersicht

3.1.5 Build-Notifiers

Bei einem fehlgeschlagenen Build kann Hudson eine E-Mail an definierte E-Mail-Adressen versenden. Plugins dieser Kategorie ermöglichen es Hudson alle Interessenten über andere Wege über den Zustand der Jobs in Kenntnis zu setzen.

Da die Standard-E-Mail von Hudson nicht konfiguriert werden kann, ist vor allem das Addon „Email-ext“³⁴ interessant. Hiermit lassen sich die Texte für Betreff und Inhalt definieren und übersichtlicher gestalten.

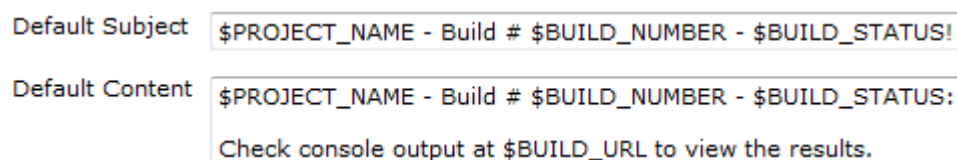


Abbildung 12: Plugin Email-ext - Globale Konfiguration

Außerdem erlaubt das Plugin je nach Zustand des Builds verschiedene Nachrichten zu versenden. Der folgende Ausschnitt zeigt die Nachricht für einen instabilen Build:

33 <http://wiki.hudson-ci.org/display/HUDSON/Release+Plugin>

34 <http://wiki.hudson-ci.org/display/HUDSON/Email-ext+plugin>

3.1.5 Build-Notifiers

Editable Email Notification

Configure email recipients, content, and what should trigger a notification

Global Recipient List

Comma-separated list of email address that should receive notifications.

Default Subject

Default Content

Trigger	Send To Recipient List	Send To Committers	Include Culprits	More Configuration
Failure	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	+ (expand)
Unstable	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	- (collapse)

Recipient List

Subject

Content

Abbildung 13: Plugin Email-ext - Job Konfiguration

Die letzte veröffentlichte Version 2.2.1 von „Email-ext“ war fehlerhaft und wurde deshalb bei der Umsetzung der Projekte nicht eingesetzt. In der Versionsverwaltung gibt es jedoch mittlerweile einen funktionsfähigen Quellcode, aus dem mit Hilfe des Build-Tools Maven in ein Plugin-Paket erstellt werden kann.

Mittlerweile existieren auch Plugins für Chat-Programme wie Twitter, Jabber und IRC, die bei fehlerhaften Builds eine Nachricht versenden.

Zu dieser Kategorie wird auch der „hudsonTracker³⁵“ gezählt. Streng genommen handelt es sich hierbei nicht um ein Plugin sondern um ein eigenständiges Programm, das über die Taskleiste des Betriebssystems auf fehlerhafte Builds hinweist. Die Informationen gewinnt das Programm dabei über die RSS-Feeds von Hudson.

3.1.6 Build-Reports

Hudson kann bereits die Testberichte des Testframeworks JUnit und die von JavaDoc³⁶ generierte Dokumentation in der Weboberfläche anzeigen.

Die Plugins dieser Kategorie erstellen eigene Berichte oder ermöglichen es die Berichte anderer Programme in der Weboberfläche darzustellen. Meistens heißen die Plugins

35 <http://wiki.hudson-ci.org/display/HUDSON/hudsonTracker>

36 <http://java.sun.com/j2se/javadoc/>

auch genauso wie die unterstützten Programme.

Dazu gehören unter anderem die Testframeworks CppUnit für C++, WebTest und Selenium für Webanwendungen, JSUnit für JavaScript und NUnit oder MSTest für .NET. Zu jedem dieser Programme gibt es bereits jeweils ein passendes Plugin, welches die generierten Berichte ausliest und in Hudsons Weboberfläche darstellt.

Bei der Umsetzung der Projekte wurde das Plugin „NUnit“ zum Auslesen der Berichte des gleichnamigen Testframeworks genutzt.

Das Plugin „xUnit“ versteht gleich die Berichte mehrerer Testframeworks, darunter befinden sich MSTest, NUnit, UnitTest++, CppUnit, BoostTest und PHPUnit. „xUnit“ selbst ist ebenfalls erweiterbar, durch weitere Plugins wird die Unterstützung noch um CppTest, JSUnit und AUnit erweitert.

Passend zu den Testframeworks gibt es auch eine Reihe von Plugins, die die Berichte von Test-Coverage-Tools auslesen und in Hudsons Weboberfläche bereitstellen können. Dazu zählen Clover, Cobertura und Emma für Java und NCover für .NET-Anwendungen. Zu dieser Kategorie gehört auch die Unterstützung vieler Metrik-Tools. Darunter befinden sich z.B. Checkstyle, FindBugs, JDepend, PMD und JMeter. Zu jedem genannten Tool gibt es jeweils ein passendes Plugin, welches die Ergebnisse der Berichte in Hudsons Weboberfläche darstellt.

Allerdings wird die Job-Konfiguration mit zunehmender Anzahl von Plugins unübersichtlicher. Um dem entgegen zu wirken, gibt es auch Plugins, die viele verschiedene Berichte auslesen können und die Konfiguration dadurch übersichtlicher gestalten.

Ein Beispiel dafür ist das Plugin „Violations“, welches auch bei der Umsetzung der Projekte eingesetzt wurde (Kapitel 4.2.2). In der Version 0.7.4 werden die Tools Checkstyle, PMD, CPD, FindBugs, pyLint, FxCop, Stylecop und Simian unterstützt. Zusätzlich lassen sich zu jedem Tool einstellen, bei welcher Anzahl von gefundenen Problemen der Build als erfolgreich oder fehlgeschlagen gekennzeichnet werden soll. In der Projekt-Übersicht werden die Ergebnisse der einzelnen Tools platzsparend in dem gleichen Graphen dargestellt, können aber auch einzeln im Detail betrachtet werden.

3.1.6 Build-Reports

In der folgenden Abbildung ist zu sehen, wie ein solcher Graph dargestellt wird:

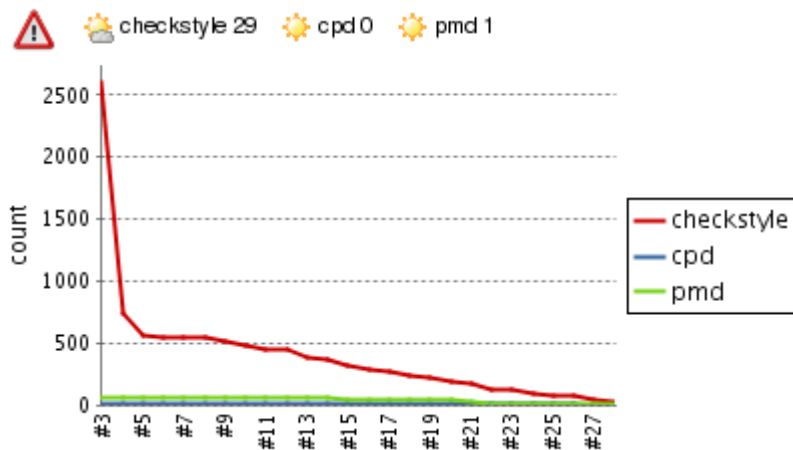


Abbildung 14: Plugin Violations - Graph

Für die Umsetzung der Projekte wurden ferner die Plugins „Warnings“ und „Task Scanner“ verwendet. Beide benötigen in der jeweils aktuellsten Version zusätzlich das Plugin „analysis-core³⁷“, in dem die Funktionen zur Darstellung der Berichte auf der Weboberfläche enthalten sind.

Das Plugin „Warnings“ ist im Stande die Warnungen und Fehlermeldungen zahlreicher Compiler zu sammeln und strukturiert darzustellen. Zu den unterstützten Compilern gehören unter anderem MSBuild, Ada (gnat), GNU (gcc), Java (javac) und JavaDoc. Die Informationen sammelt das Plugin aus den Log-Dateien des Builds, die in der Job-Konfiguration angegeben werden können.

37 <http://wiki.hudson-ci.org/display/HUDSON/Static+Code+Analysis+Plug-ins>

Die folgende Abbildung zeigt die Konfiguration zum Auslesen von Hudsons Konsolenausgabe nach Warnungen von MSBuild:

Suche nach Compiler Warnungen

Konsole durchsuchen

Falls die Konsolenausgabe nach Warnungen durchsucht werden soll, muss diese Option aktiviert sein.

Dateiauswahl

Angabe einer [ANT Fileset includes](#) Anweisung, die den Pfad zu den Dateien angibt, in denen nach Warnungen gesucht werden soll. Falls das Feld leer bleibt, wird nur die Konsolenausgabe nach Warnungen durchsucht.

Parser

Diese Parser werden zum Parsen der Konsolenausgabe bzw. der angegebenen Dateien verwendet. Wird kein Parser ausgewählt, so laufen immer alle Parser mit (benötigt mehr Speicher und Rechenzeit).

Abbildung 15: Plugin Warnings - Job Konfiguration

In den erweiterten Einstellungen kann das Verhalten des Plugins noch weiter definiert werden, was in der folgenden Abbildung zu sehen ist:

Immer aktivieren

Normalerweise wird dieses Plug-in nur nach erfolgreichen oder instabilen Builds ausgeführt. Falls das Plug-in auch nach fehlgeschlagenen Builds ausgeführt werden soll, muss diese Checkbox aktiviert werden.

Zu berücksichtigende Warnungen

Angabe einer [ANT Fileset includes](#) Anweisung, die die Warnungen festlegt, die im Warnungsbericht erscheinen sollen (basierend auf deren Dateinamen).

Zu ignorierende Warnungen


Angabe einer [ANT Fileset excludes](#) Anweisung, die die Warnungen festlegt, die nicht im Warnungsbericht erscheinen sollen (basierend auf deren Dateinamen).

Abbildung 16: Plugin Warnings - Erweiterte Job Konfiguration

3.1.6 Build-Reports

Zu den erweiterten Einstellungen gehört auch die Definition des Build-Status. Hier wird festgelegt, ab wann ein Build als instabil oder fehlgeschlagen gilt.

Build Status



 Gesamt Neue Gesamt Neue

Grenzwerte:

Konfiguriert den Status und Gesundheitszustand eines Builds. Ein Build wird als instabil bzw. fehlgeschlagen markiert, wenn die Gesamtanzahl oder die Anzahl der neuen Warnungen den festgelegten Grenzwert überschreiten. Der Gesundheitszustand wird genauso über Grenzwerte bestimmt. Falls die Anzahl der Warnungen zwischen diesen beiden Werten liegt, so wird sie interpoliert.

Prioritäten Auswahl

Nur Priorität hoch Prioritäten hoch und normal
 Alle Prioritäten

Legt die Prioritäten fest, die bei der Berechnung des Build Status und Gesundheitszustands herangezogen werden.

Abbildung 17: Plugin Warnings - Erweiterte Job Konfiguration (Fortsetzung)

Unter der den einzelnen Builds erhält man durch das Plugin eine Zusammenfassung der Warnungen. Dabei wird der aktuelle mit dem vorherigen Build verglichen und neue oder behobene Warnungen erkannt (Abb. 18).

Vergleich mit letzter Analyse

Alle Warnungen	Neue Warnungen	Behobene Warnungen
18	<u>1</u>	0

Zusammenfassung

Gesamt	Hohe Priorität	Normale Priorität	Niedrige Priorität
18	0	<u>18</u>	0

Details

Datei	Gesamt	Verteilung
ConfigPrintUserControl.vb	1	<div style="width: 10%; background-color: yellow;"></div>
FilterControl.ascx.vb	4	<div style="width: 40%; background-color: yellow;"></div>
FilterHelper.vb	5	<div style="width: 50%; background-color: yellow;"></div>
MeasDataView.aspx.vb	3	<div style="width: 30%; background-color: yellow;"></div>
NotifierTest.vb	2	<div style="width: 20%; background-color: yellow;"></div>
UsermanagementReport.aspx.vb	3	<div style="width: 30%; background-color: yellow;"></div>

Abbildung 18: Plugin Warnings - Build Zusammenfassung

Die Gründe für die Warnungen können unter dem Reiter „Details“ nachgelesen werden. Sogar den bemängelten Quellcode kann man sich im Browser anzeigen lassen und so

direkt das Problem nachvollziehen.

In der Projekt-Übersicht wird aus den gesammelten Daten ein Graph generiert, der die Anzahl der Warnungen über die Builds darstellt. Die Farbe stellt dabei die Priorität der Warnung dar (blau = niedrig, gelb = normal und rot = hoch):

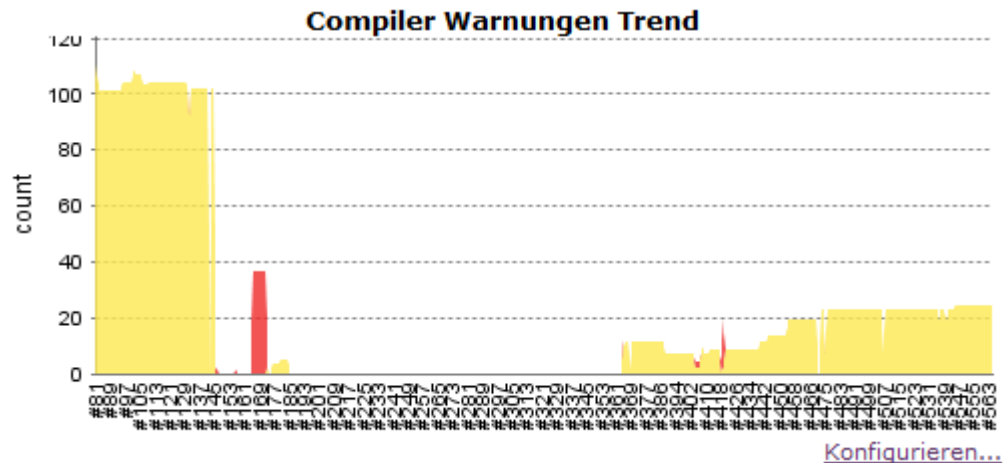


Abbildung 19: Plugin Warnings - Projekt-Graph

Das Plugin „Task Scanner“ durchsucht den vorgegebenen Quellcode nach Kommentarzeilen, die offene Punkte in der Implementierung des Codes kennzeichnen. Die Kennzeichnung und Priorität der offenen Punkte kann in der Job-Konfiguration definiert werden, was in der folgenden Abbildung eingesehen werden kann:

Suche im Arbeitsbereich nach offenen Punkten

Zu untersuchende Dateien
 Angabe einer [ANT Fileset includes](#) Anweisung, die den Pfad der zu untersuchenden Dateien bestimmt, z.B. `**/*.java`. Als Ausgangsverzeichnis für diese Anweisung wird der [Arbeitsbereich](#) verwendet. Falls kein Wert eingetragen wird, dann wird die Vorgabe `**/*.java` benutzt.

Zu ignorierende Dateien
 Angabe einer [ANT Fileset excludes](#) Anweisung, die die Dateien angibt, die beim Scannen nicht berücksichtigt werden sollen, z.B. Dateien von Fremdbibliotheken. Als Ausgangsverzeichnis für diese Anweisung wird der [Arbeitsbereich](#) verwendet.

Kennzeichnung offener Punkte

Hohe Priorität	Normale Priorität	Niedrige Priorität	Groß-/Kleinschreibung ignorieren
<input type="text" value="FIXME, HA"/>	<input type="text" value="TODO, TC"/>	<input type="text" value="REVIEW, I"/>	<input checked="" type="checkbox"/>

Konfiguriere die Texte, nach denen in den angegebenen Dateien gesucht werden soll. Für jede Priorität kann eine durch Kommas getrennte Liste von Texten definiert werden, z.B. `TODO, FIXME`, o.a. Die Groß- bzw. Kleinschreibung kann dabei optional ignoriert werden.

Abbildung 20: Plugin Task Scanner - Job Konfiguration

3.1.6 Build-Reports

Die Ergebnisse der Suche werden genauso wie bei dem Plugin „Warnings“ in der Projekt-Übersicht und in den einzelnen Builds dargestellt.

Ein weiteres nützliches Plugin nennt sich „Disk Usage“. Dieses liefert Informationen über den Speicherplatzverbrauch der Jobs auf der Festplatte. Dadurch lässt sich der künftige Platzbedarf der Jobs besser einschätzen und man kann erkennen, welche Jobs bei Platzmangel gesichert und aufgeräumt werden sollten.

In der Projekt-Übersicht wird die Entwicklung des Platzbedarfs über die Builds hinweg in einem Graph dargestellt, was sehr gut in Abbildung 21 ersichtlich wird.

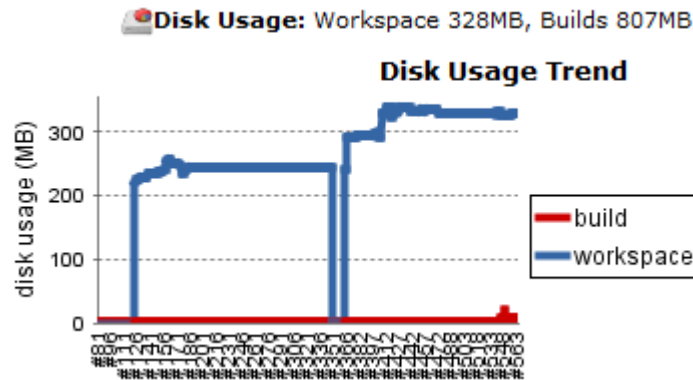


Abbildung 21: Plugin Disk Usage - Projekt Übersicht

Build-Verlauf	(Trend)
#565 22.12.2009 16:01:04 8MB	
#564 22.12.2009 15:16:04 8MB	

Abbildung 22: Plugin Disk Usage - Build-Verlauf

Im Build-Verlauf (Abb. 22) wird rechts neben den Einträgen der Platzverbrauch der einzelnen Builds angezeigt

Eine Gesamt-Übersicht über den Platzverbrauch aller Projekte erhält man im Verwaltungsbereich, dargestellt in Abb. 23. Das Plugin aktualisiert die Werte alle 15 Minuten, bei Bedarf kann die Messung in dieser Übersicht aber auch sofort ausgelöst werden.

Builds: 933MB Workspace: 1GB

Project name	Builds	Workspace
[REDACTED]	815MB	328MB
[REDACTED]	95MB	341MB
[REDACTED]	2MB	215MB
[REDACTED]	19MB	107MB
[REDACTED]	736KB	52MB
[REDACTED]	466KB	51MB
[REDACTED]	850KB	13KB

Disk usage is calculated each minutes. If you want to trigger the calculation now, click on the button.

Record Disk Usage

Abbildung 23: Plugin Disk Usage - Gesamt-Übersicht

3.1.7 Artifact-Uploaders

In Hudson kann man einen Job so konfigurieren, dass bestimmte Dateien nach einen

Build archiviert werden. Diese sogenannten Artefakte werden dann in dem jeweiligen Build-Verzeichnis abgelegt. Plugins dieser Kategorie bieten weitere Möglichkeiten an diese Artefakte zu verteilen.

Beispiele dafür sind die Plugins „FTP-Publisher“ und „SCP“, mit denen Dateien per FTP bzw. SFTP kopiert werden können. Allerdings ist der „FTP-Publisher“ in der aktuellen Version 0.9 fehlerhaft und sollte nicht verwendet werden. Als Alternative würde es sich anbieten, in der Job-Konfiguration einen Build-Schritt hinzuzufügen und die gewünschten Dateien mit einem beliebigen FTP-Programm über die Kommandozeile zu kopieren.

Hat man vor Hudson für die Öffentlichkeit oder bestimmte Kunden frei zugänglich zu machen, könnte das Plugin „Build Publisher“ interessant sein. Damit lassen sich die Ergebnisse eines Builds auf einem weiteren Hudson-Server veröffentlichen. So könnte man z.B. einen privaten Hudson für die Entwickler einrichten, der durch eine Firewall geschützt ist und nur die erfolgreichen Builds auf einen weiteren öffentlichen Hudson zugänglich machen.

3.1.8 External site/tool integrations

Durch zahlreiche Plugins wird die Zusammenarbeit von Hudson mit weiteren Programmen verbessert, die bei der Softwareentwicklung eingesetzt werden.

Dazu gehören insbesondere Systeme, mit deren Hilfe die Entwickler Änderungen und Fehler in ihrer Software dokumentieren und zurückverfolgen können. Durch die Plugins für die Systeme Bugzilla, Google Code, JIRA, Mantis Bug Tracker und Trac ist es möglich in den Build-Berichten von Hudson auf die betreffenden Einträge zu verweisen.

Bei der Umsetzung der Projekte wurde das Plugin „Mantis“ eingesetzt, um auf vorhandene Bug Tracker-Einträge der Projekte zu verweisen. Das Plugin durchsucht dazu die Kommentare aus den einzelnen Builds nach bestimmten Text-Mustern und ersetzt diese mit einem Verweis auf die Einträge. Dieses Muster muss in der Job-Konfiguration festgelegt werden. Bei eWorks ist beispielsweise das Muster "#%ID%" üblich. Der Text #123 würde dann durch einen Verweis auf den Bug Tracker-Eintrag mit der ID 123 ersetzt. Durch eine weitere Option in der Job-Konfiguration kann man dem

3.1.8 External site/tool integrations

Plugin zusätzlich erlauben, den referenzierten Einträgen Kommentare hinzuzufügen. Das Plugin notiert darin den Verweis auf den Build im Hudson, durch wen er ausgelöst wurde und den darin enthaltenen Kommentar.

Ein weiteres nützliches Tool für Hudson ist das Firefox-Addon „Hudson Build Monitor“. Nach der Installation integriert es sich in die untere Statusleiste von Firefox und kann mit Hilfe von Hudsons RSS-Feeds den Status der letzten Builds anzeigen. Dazu klickt man den gewünschten RSS-Feed mit der rechten Maustaste an und fügt ihn über die Option „Add Link to Hudson Build Monitor“ hinzu. Hält man nun die Maus über die Einträge in der Statusleiste, sieht man eine Liste der letzten 10 Builds. Klickt man mit der linken Maustaste auf den Eintrag, erhält man eine Liste mit Verweisen auf die Builds.

In der folgenden Abbildung ist die Darstellung der RSS-Feeds im Mozilla-Firefox zu sehen:

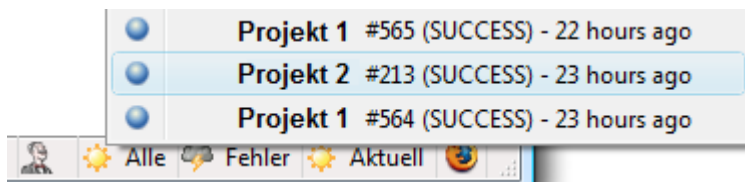


Abbildung 24: Firefox Addon Hudson Build Monitor

3.1.9 UI plugins

Plugins dieser Kategorie verändern die Weboberfläche von Hudson.

In Hudson werden normalerweise erfolgreiche Builds mit einem blauen und fehlgeschlagene Builds mit einem roten Kreis gekennzeichnet. Wer für erfolgreiche Builds einen grünen Kreis bevorzugt, dem ist mit dem Addon „Green Balls“ geholfen.

Das Plugin „DocLinks“ fügt der Projekt-Übersicht Verweise auf Dokumente hinzu, die in der Job-Konfiguration definiert werden können. Wird in den Builds beispielsweise für den Quellcode eine Dokumentation erstellt, könnte man mit dem Plugin immer auf die aktuellste Version verweisen.

Auf der Hauptseite von Hudsons Weboberfläche lassen sich weitere Ansichten anlegen und so konfigurieren, dass nur bestimmte Jobs angezeigt werden. Mit dem Plugin „Sectioned View“ lassen sich diese Ansichten noch vielfältiger gestalten. So ist es nun mit Hilfe dieses Plugins möglich, eigene Text-Bereiche oder die Graphen mit den Build-

Ergebnissen hinzuzufügen und genauer festzulegen, wie die Ansicht aufgebaut sein soll. Diese erweiterten Ansichten stehen für private und öffentliche Ansichten zur Verfügung. Bei steigender Anzahl von Jobs kann dieses Plugin sehr nützlich sein, um den Überblick über den Zustand der Jobs zu behalten.

Das Plugin „Dashboard View“ bietet ähnliche Optionen, um die Ansichten in Hudson zu gestalten. Darüber hinaus bietet es weitere Arten von Tabellen (z.B. Test-Statistiken) und Graphen (z.B. Kreisdiagramm oder eine Aufsummierung von Testergebnissen pro Tag) an, um den Zustand der Jobs darzustellen.

3.1.10 Authentication and user management

Zu dieser Kategorie gehören Plugins, die sich mit der Verwaltung und Authentifizierung von Benutzerdaten befassen.

Mit dem „LDAP Email“ Plugin können die E-Mail-Adressen der Hudson-Benutzer aus einem LDAP-Verzeichnis ausgelesen werden.

Die Authentifizierung der Benutzer kann durch das „Active Directory“-Plugin auch durch Active Directory geschehen, statt über Hudsons eigene Benutzerverwaltung.

Die Konfiguration der Jobs kann von einem Benutzer normalerweise nur mit Konfigurations-Rechten eingesehen und verändert werden. Mit dem Plugin „Extended Read Permission“ wird den Benutzerrechten eine weitere Option hinzugefügt, mit der Leserechte für die Job-Konfiguration vergeben werden können. Dies könnte z.B. beim Anlegen neuer Jobs hilfreich sein, wenn ein Benutzer wissen möchte, wie andere Jobs umgesetzt wurden.

3.1.11 Verschiedenes

In der letzten Kategorie werden alle restlichen Plugins zusammengefasst.

Hier finden sich weitere nützliche Erweiterungen, die auch bei der Umsetzung der Projekte eingesetzt wurden. Dazu gehört die Plugins „Backup“, „Change Log History“ und „The Continuous Integration Game“.

Mit „Backup“ lässt sich die Hudson-Installation mitsamt aller Jobs zur Datensicherung speichern und bei Bedarf auch wieder herstellen. Die Konfiguration des Plugins kann im Verwaltungs-Bereich von Hudson vorgenommen werden. Dabei lässt sich neben dem

3.1.11 Verschiedenes

Dateinamen und Format des Backups auch einstellen, welche Inhalte der Jobs gesichert werden sollen. Zur Auswahl stehen das Arbeitsverzeichnis, die Build-Verzeichnisse, die Artefakte und die gespeicherten MD5 Prüfsummen (fingerprints).

Werden in den Jobs einzelne Builds gelöscht, gehen die darin enthaltenen Kommentare zu den Änderungen aus der Versionsverwaltung leider verloren. Mit „Change Log History“ werden diese vor dem Löschen an die Kommentare des folgenden Builds angehängt. So bleiben alle Kommentare für die Übersicht „Änderungen“ in der einzelnen Jobs erhalten.

Ein sehr interessantes Plugin ist „The Continuous Integration Game“. Diese Erweiterung verfolgt einen spielerischen Ansatz, um die Motivation der Entwickler zu erhöhen, in kürzeren Abständen funktionierenden und qualitativ guten Quellcode in der Versionsverwaltung einzuchecken und neue Tests hinzuzufügen. Nach einem Build bekommen die Entwickler, die ihn ausgelöst haben, Punkte gutgeschrieben. Für erfolgreiche Builds erhält man einen Punkt und fehlgeschlagene Builds kosten dem Verursacher 10 Punkte. Für jeden neuen Test erhält man einen Punkt und für jeden Test, der seit dem letzten Build fehlschlägt, wird 1 Punkt abgezogen.

Die Regeln für die Punktevergabe lassen sich aktuell (Version 1.15) nur durch Anpassung des Quellcodes verändern. Andere Plugins können durch Implementierung des Interfaces „Rule“ ihre eigenen Regeln definieren und zur Bewertung des Builds beitragen. Nach der Definition aller Regeln müssen diese noch in einem „RuleSet“ zusammengefasst und an die Regeln des Spiels übergeben werden. Im folgenden Quellcode-Abschnitt ist zu sehen, wie die Übergabe der Regeln aussehen könnte:

```
[1] hudson.plugins.cigame.PluginImpl.GAME_PUBLISHER_DESCRIPTOR
    .getRuleBook().addRuleSet(pluginruleset);
```

Text 2: Plugin CI-Game - Hinzufügen von Regeln

Bestimmte Plugins liefern bereits eigene Regeln mit. Dazu gehören Warnings, Task Scanner, Violations, PMD, FindBugs und Checkstyle. Für behobene Probleme vergeben die Plugins abhängig von Schwere-Grad 1 bis 5 Punkte oder ziehen genauso viele Punkte ab, wenn es sich um ein neues Problem handelt.

In der Zusammenfassung eines Builds notiert das Spiel wie viel Punkte dafür vergeben

wurden.

Der enthaltene Verweis listet die genaue Bewertung auf und welche Benutzer die Punkte bekommen.



The build was worth [-5 points](#)

Abbildung 25: Plugin CI-Game - Build

Score card

Rule	Points ↑
The build was successful	1.0
1 new compiler warnings were found	-1.0
1 new open HIGH priority tasks were found	-5.0

Abbildung 26: Plugin CI-Game - Build Details - Punkte

Participating players ↓

█

Abbildung 27: Plugin CI-Game - Build Details - Benutzer

Auf der Hauptseite der Weboberfläche führt ein Verweis auf die Rangliste der Benutzer für alle Projekte. Eine Punkteliste auf Projekt-Ebene gibt es in der aktuellen Version des Plugins leider nicht. In der folgenden Abbildung ist das Leader Board zu sehen, welches einen Gesamtüberblick der Punkte der eingerichteten Projekte (siehe Kapitel 4.2) liefert:

Leader board

Participant	Description	Score ↑
█		314.0
█		15.0
█		10.0
█		5.0
█		4.0
█		0.0
█		0.0
█		-28.0
█		-40.0

Abbildung 28: Plugin CI-Game - Rangliste

3.2 Verwaltung der Plugins

Die Installation von Plugins lässt sich am einfachsten über Hudsons Weboberfläche durchführen. Alternativ kann man die entsprechende HPI-Datei auch einfach in den Plugin-Ordner `HUDSON_HOME\plugins` kopieren und Hudson (bzw. Tomcat) neu starten.



[Plugins verwalten](#)
[Plugins installieren](#)

Abbildung 29: Link zur Plugin-Verwaltung

Damit man bei der großen Palette von Plugins nicht den Überblick verliert, gibt es im Verwaltungs-Bereich von Hudsons Weboberfläche eine Sektion, die sich ausschließlich mit der Verwaltung der Plugins befasst.

3.2 Verwaltung der Plugins

Die Plugin-Verwaltung unterteilt sich in 4 Bereiche. Im ersten Bereich wird man auf neue Versionen bereits installierter Plugins hingewiesen und kann diese mit einem Mausklick aktualisieren. Die neue Version wird automatisch heruntergeladen und installiert, für die Aktivierung ist allerdings ein Neustart von Hudson erforderlich.

Aktualisierungen			
Verfügbar			
Installiert			
Erweiterte Einstellungen			
Installieren	Name ↓	Version	Installiert
<input type="checkbox"/>	Doxygen Plugin This plugin publishes HTML reports generated by the Doxygen tool.	0.5	0.4
<input type="checkbox"/>	Green Balls Changes Hudson to use green balls instead of blue for successful builds	1.5	1.4

Abbildung 30: Aktualisierung von Plugins

Der zweite Bereich dient zur Installation von Plugins in ihrer aktuellsten Version. Alle in dem Update-Center verfügbaren Plugins werden hier mit ihren Namen, Versionsnummer und kurzer Beschreibung aufgeführt. Seit der Hudson Version 1.337 werden die Plugins zur besseren Übersicht in Kategorien zusammengefasst. Jeder Eintrag verweist außerdem auf die Wiki-Seite des Plugins, auf der jeder Autor weitere Informationen dazu veröffentlichen kann. Der folgende Bildausschnitt zeigt den ersten Eintrag der Liste:

Aktualisierungen		
Verfügbar		
Installiert		
Erweiterte Einstellungen		
Installieren ↓	Name	Version
Artifact Uploaders		
<input type="checkbox"/>	Artifactory Plugin This plugin allows deploying maven artifacts and build info to Artifactory.	1.0.1

Abbildung 31: Verfügbare Plugins

Im dritten Bereich erhält man eine Liste der bereits installierten Plugins (Abb. 32). Darin werden der Name, die Versionsnummer und eine kurze Beschreibung aufgeführt. In diesen Bereich kann man außerdem auswählen, welche Plugins beim Start von Hudson geladen werden sollen. Möchte man eine Erweiterung deaktivieren, entfernt

Aktualisierungen		
Verfügbar		
Installiert		
Erweiterte Einstellungen		
Aktiviert	Name ↓	Version
<input checked="" type="checkbox"/>	Hudson Backup plugin Backup or Restore your hudson configuration files	1.3

Abbildung 32: Installierte Plugins

man einfach davor den Haken und startet Hudson neu.

Das Entfernen von Plugins ist über die Weboberfläche nicht möglich. Dazu muss Hudson gestoppt und das Plugin per Hand gelöscht werden. Dabei sind die Datei `HUDSON_HOME\plugins\<Plugin-ID>.hpi` und das Verzeichnis `HUDSON_HOME\plugins\<Plugin-ID>` zu entfernen. Die Plugin-ID ist üblicherweise eine Kurzfassung des Plugin-Namens.

Im letzten Bereich namens „Erweiterte Einstellungen“ lassen sich unter anderem auch Plugins durch Übergabe der entsprechenden HPI-Datei installieren. Dadurch ist es möglich bestimmte Versionen oder unveröffentlichte Plugins über die Weboberfläche zu installieren. Falls das Update-Center nicht erreichbar ist (z.B. aufgrund einer Firewall), kann man hier bei Bedarf die Zugangsdaten für einen Proxy-Server einstellen, der die Anfragen an das Update-Center weiterleiten kann.

Aktualisierungen Verfügbar Installiert **Erweiterte Einstellungen**

HTTP-Proxy Konfiguration

Server ?

Port ?

Benutzername ?

Kennwort

Plugin hochladen

Geben Sie hier den Pfad zu einer lokalen Plugin-Datei (Dateiendung .hpi) an, die installiert werden soll.

Datei:

Letzte Datenaktualisierung des Update-Centers: Vor 8 Stunden 24 Minuten

Abbildung 33: Erweiterte Einstellungen der Plugin-Verwaltung

3.3 Plugin-Dateistruktur (HPI)

Eine HPI-Datei ist eigentlich ein Java-Archiv³⁸ (kurz: JAR), das jedoch bestimmten Konventionen³⁹ folgt. Dazu gehört unter anderem folgende Verzeichnisstruktur:

```
beispiel.hpi
+- META-INF
|   +- MANIFEST.MF
+- WEB-INF
|   +- classes
|   +- lib
+- (Statische Dateien)
```

Text 3: Plugin Verzeichnisstruktur

Der Dateiname ohne die Endung .hpi stellt die Plugin-ID dar und ist einzigartig unter allen Plugins. Üblicherweise ist die Plugin-ID die Kurzform des Plugin-Namens.

Die Datei MANIFEST.MF enthält zusätzlich Informationen über den vollständigen Namen der Plugin-Klasse, den Namen des Plugins und die Abhängigkeiten zu anderen Plugins.

Zu den statischen Dateien gehören z.B. Bilder, HTML-Dateien, CSS-Stylesheets und JavaScript-Dateien.

³⁸ http://de.wikipedia.org/wiki/Java_Archive

³⁹ Plugin-Dateistruktur: <http://wiki.hudson-ci.org/display/HUDSON/Plugin+structure>

4 Integration des Continuous Integration Servers

Nachdem Hudson als Sieger der Evaluation feststand, ging es nun darum den CIS für ausgewählte Projekte der Firma eWorks GmbH einzusetzen.

Der CIS sollte dazu in einem virtuellen Rechner laufen und auf einem Server von eWorks installiert werden. Für die Virtualisierung von Rechnern setzte eWorks die Anwendung „VMWare Server⁴⁰“ ein, welche es erlaubt die Ressourcen des physischen Servers auf die virtuellen Rechner zu verteilen. Die virtuellen Rechner haben gegenüber den physischen gewisse Vorteile, die vor allem in kleineren Unternehmen eine Rolle spielen. Der Verwaltungsaufwand von virtuellen Rechnern ist geringer, da sie mit Hilfe einer grafischen Oberfläche von einem anderen Rechner aus gestartet, gestoppt und die Ressourcen wie Arbeitsspeicher und Festplattenspeicher den Bedürfnissen angepasst werden können. Außerdem kann man durch die Virtualisierung die Anschaffung weiterer physischer Rechner einsparen und benötigt dadurch auch keine größeren Räume zur Unterbringung der Server. Die Systemumgebung eines einzelnen Rechners wird als VMWare Image in Dateien gekapselt und von einem VMWare Server verwaltet. Dies vereinfacht auch die Datensicherung der Rechner, da von den virtuellen Rechner jederzeit ein Image existiert und nicht wie bei physischen Rechnern erst erstellt werden muss. Nach außen hin verhalten sich die virtuellen genauso wie physische Rechner, sie besitzen z.B. einen eigenen Namen und eine eigene IP-Adresse und der Datenaustausch zwischen den Rechnern geschieht ebenfalls über Netzwerkverbindungen.

4.1 Aufbau des virtuellen Rechners

Bei eWorks wird VMWare häufig für die Entwicklung und das Testen von Software eingesetzt. Die Entwickler haben dazu eine Vielzahl von VMWare Images mit unterschiedlichen Betriebssystemen erstellt, um als Ausgangsbasis für neue virtuelle Rechner zu dienen.

Als Basis für den den CIS wurde ein VMWare Image mit dem Betriebssystem MS Windows Server 2008 gewählt, da in dem Unternehmen die Anwendungen fast ausschließlich unter Windows entwickelt werden und die Mitarbeiter große Erfahrung

40 <http://www.vmware.com/de/products/server/>

mit dem Umgang und der Konfiguration des Betriebssystems besitzen. Um die Einrichtung des Rechners auf dem Server zu beschleunigen, wurde das Image im Vorfeld auf einem Arbeitsrechner vorbereitet. Dabei wurden unter anderem das Betriebssystem aktualisiert, Hudson installiert und konfiguriert und die Ressourcen für den virtuellen Rechner eingestellt. Damit ausreichend Platz für die Installation weiterer Software und die Durchführung der Builds zur Verfügung steht, wurde der verfügbare Festplattenspeicher des virtuellen Rechners auf 40GB festgelegt. Als Ausgangsgröße für den Arbeitsspeicher wurden 3GB zur Verfügung gestellt. Sollten diese Ressourcen im Laufe des Betriebs knapp werden, lassen sich die Größen auch nach der Installation auf dem Server noch weiter anpassen.

4.1.1 Hudson Installation

Als Java-Anwendung benötigt Hudson zum Betrieb eine Java Laufzeit-Umgebung (englisch: Java Runtime Environment). Deshalb wurde zur Vorbereitung JDK (englisch: Java Development Kit) 1.6 installiert, welches neben Entwicklungswerkzeugen für Java ebenfalls die benötigte Java Runtime Environment (kurz: JRE) enthält.

Hudson kann innerhalb eines Java Servlet Containers (wie z.B. Tomcat⁴¹) oder als eigenständige Anwendung (englisch: standalone) betrieben werden. Bei der Standalone-Variante arbeitet Hudson mit dem integrierten Servlet Container Winstone. Da für zukünftige Java-Projekte eventuell sowieso ein Servlet Container benötigt wird, wurde Hudson innerhalb von Tomcat installiert. Außerdem lässt sich Hudson mit Tomcats „Web Application Manager“⁴² bei Bedarf einfach starten und stoppen. Die Installation von Tomcat erfolgte mit Hilfe einer Version mit integriertem Windows Service Installer. Dabei wurde die zu dem Zeitpunkt aktuellste Version 6.0.20⁴³ verwendet und im Verzeichnis „*c:\tomcat6*“ installiert. Besondere Einstellungen bei der Installation waren nicht notwendig.

Die Installation von Hudson selbst gestaltet sich sehr einfach. Dazu wird dessen Web-Archiv Datei „*hudson.war*“ in den Applikations-Ordner von Tomcat kopiert, welcher sich unter „*C:\Tomcat6\webapps*“ befindet. Nach dem Kopieren installiert sich Hudson selbstständig unter Tomcat.

41 <http://tomcat.apache.org>

42 <http://tomcat.apache.org/tomcat-6.0-doc/manager-howto.html>

43 <http://apache.easy-webs.de/tomcat/tomcat-6/v6.0.20/bin/apache-tomcat-6.0.20.exe>

Die Konfigurationsdateien von Hudson und die Dateien aller verwalteten Projekte werden nicht in den Verzeichnissen von Tomcat abgelegt, sondern befinden sich in einem eigenen Verzeichnis. Die Standardeinstellung für dieses Verzeichnis lautet „*c:\hudson*“. Über die Umgebungsvariable namens *HUDSON_HOME* lässt sich diese Einstellung auch ändern. Für die Installation von Hudson wurde das Standardverzeichnis beibehalten.

4.1.2 Tomcat Konfiguration

Für den Betrieb bei eWorks sind an Tomcat weitere Einstellungen nötig. Dazu zählen die Einstellung der URL für Hudsons Weboberfläche und die Konfiguration der Java Virtual Maschine.

4.1.2.1 Umstellung der Hudson URL

Die Weboberfläche von Hudson lässt sich nach der Installation lokal unter der URL „*http://localhost:8080/hudson*“ abrufen. Diese lässt sich auch noch weiter vereinfachen, um die Benutzbarkeit zu erhöhen.

Der Port lässt sich über eine Einstellung in der Tomcat-Konfigurationsdatei „*C:\Tomcat6\conf\server.xml*“ ändern. Um den Port beispielsweise von 8080 auf 80 zu ändern ist der Connector wie folgt anzupassen:

```
[1] <Connector port="80"  
    protocol="HTTP/1.1"  
[2]     connectionTimeout="20000"  
[3]     redirectPort="8443"  
[4]     URIEncoding="UTF-8" />
```

Text 4: Tomcat Port-Einstellung

Damit die Weboberfläche im Root-Verzeichnis ohne den Zusatz „*/hudson*“ erreicht werden kann, ist die einfachste Methode die Datei „*hudson.war*“ vor dem Kopieren in den Tomcat Applikationsordner in „*ROOT.war*“ umzubenennen. Dadurch wird Hudson automatisch im ROOT-Verzeichnis von Tomcat installiert und ist lokal unter der URL „*http://localhost*“ erreichbar.

4.1.2.2 Konfiguration der JVM

Als Java-Anwendung wird Tomcat mit Hilfe einer JVM (englisch: Java Virtual

4.1.2.2 Konfiguration der JVM

Maschine) ausgeführt. Diese JVM reserviert für sich einen Teil des Arbeitsspeichers. Für jeden neuen Java-Thread wird nicht nur in der JVM ein Objekt angelegt, sondern auch im Betriebssystem ein Thread erstellt. Dadurch schrumpft der verfügbare Arbeitsspeicher mit jedem neuen Java-Thread zusätzlich. Damit für den dauerhaften Betrieb von Hudson ausreichend Arbeitsspeicher zu Verfügung steht, sollten die Einstellungen für die JVM an die Größe des Arbeitsspeichers angepasst werden. Die folgenden Einstellungen sind geeignet für 3GB Arbeitsspeicher.

Da Hudson innerhalb des Servlet Containers Tomcat läuft, sind die Einstellungen für die JVM dort vorzunehmen. Zu diesem Zweck besitzt Tomcat ein Konfigurationsprogramm, welches man über das Startmenü in Windows aufrufen kann. Ist das Programm gestartet, können unter dem Reiter „Java“ alle notwendigen Einstellungen vorgenommen werden.

In dem Feld Java Options wird durch die folgende Zeile der Parameter *MaxPermSize* festgelegt:

```
[1] -  
XX:MaxPermSize=128m
```

Text 5: Tomcat JVM-Einstellung

Mit diesem Parameter setzt man die Maximalgröße des Abschnitts des Heap-Speichers fest, der für die permanente Objektgeneration reserviert ist und alle Reflexionsdaten für die JVM enthält. Die Standardgröße für den Parameter ist 64m, wobei das m für die Maßeinheit MB (Megabyte) steht. Diese Größe sollte erhöht werden, um die Leistungen von Anwendungen zu optimieren, die dynamisch viele Klassen laden und entladen. Kommt es beim Betrieb von Tomcat zu der Ausnahme „java.lang.OutOfMemoryError: PermGen space“ muss der Wert erhöht werden.

Außerdem wurden folgende Optionen wie folgt angepasst:

Initial memory pool (MB): 64 (Standardwert 50)
Maximum memory pool (MB): 256 (Standardwert 256)
Thread stack size (KB): 1024 (Standardwert 512)

Initial und *Maximum memory pool* steuert die Anfangs- und Maximalgröße des Java-Heap-Speichers in MB. Durch eine ordnungsgemäße Einstellung dieser Parameter kann der Aufwand für die Garbage-Collection reduziert werden, um damit die Antwortzeit und den Durchsatz des Servers zu verbessern. Jeder Java Thread besitzt einen Java und

einen C Stack. Mit der Variablen *Thread stack size* lässt sich die Größe des C Stacks einstellen. Kommt es beim Betrieb von Tomcat zu der Ausnahme „OutOfMemoryError: unable to create new native thread“ müssen die Werte verringert werden, damit der Rechner genug Arbeitsspeicher zum Anlegen der C Stacks zur Verfügung hat.

4.1.2.3 Windows Service Einstellungen

Tomcat wird bei der Installation unter Windows Server 2008 als Service eingerichtet und wird automatisch beim Hochfahren des Betriebssystems gestartet. Aus Sicherheitsgründen sollte dieser Service mit einem anderen Benutzer als dem Administrator gestartet werden. Dazu wurde zuerst ein Benutzer namens „cis“ angelegt und danach der Tomcat Service so konfiguriert, dass er automatisch beim Hochfahren des Betriebssystems von diesem Benutzer gestartet wird.

4.2 Integration der Projekte

Die Integration mehrerer aktueller Projekte der Firma soll aufzeigen, wie mit Hilfe von Hudson deren CI-Prozesse umgesetzt werden können. Durch den frühzeitigen Einsatz von Hudson werden außerdem wichtige Informationen für die abschließende Beurteilung des CIS-Einsatzes aufgezeichnet. Zu den Aufzeichnungen gehören beispielsweise die Anzahl fehlgeschlagener Builds und die Testergebnisse. Die Einbindung einzelner Mitarbeiter in die CI-Prozesse von Hudson liefert zusätzlich Rückmeldungen über die Benutzerfreundlichkeit (z.B. bei der Einrichtung neuer Projekte oder Verwendung der Oberfläche) des CIS.

4.2.1 Projektauswahl

Bei der Auswahl handelt es sich um Projekte, die während des Zeitraums der Diplomarbeit bei eWorks durchgeführt wurden und typischen Projekten der Firma entsprechen.

Die Projekte für den CIS wurden unter anderem nach der Art der eingesetzten Technologien (wie z.B. die Programmiersprache), die Art der Anwendung (wie Desktop- oder Web-Anwendung) und dem Projekttyp (Support, Weiterentwicklung und Neuentwicklung einer Anwendung) ausgewählt.

In den folgenden Kapiteln werden die ausgewählten Projekte genauer vorgestellt und

4.2.1 Projektauswahl

die bisherigen Vorgänge bei der Entwicklung der Anwendungen erläutert. Im Anschluss folgt die Umsetzung der CI-Prozesse in Hudson. Die Projektnamen wurden auf Wunsch der Firma durch allgemeine Begriffe ersetzt.

4.2.2 Allgemeines zur Umsetzung in Hudson

4.2.2.1.1 Jobs

In Hudson werden die Projekte durch sogenannte „Jobs“ umgesetzt. Darin sind alle notwendigen Informationen zur Durchführung der Build-Prozesse enthalten. Die genaue Konfiguration der Jobs wird in der Umsetzung der einzelnen Projekte beschrieben.

Neue Jobs lassen sich durch den Verweis „Neuen Job anlegen“ auf der Hauptseite von Hudsons Weboberfläche erstellen. Dabei werden verschiedene Optionen angeboten und wie folgt beschrieben:

Externer Job überwachen

Dieses Profil erlaubt die Überwachung von Prozessen, die außerhalb von Hudson ausgeführt werden - eventuell sogar auf einem anderen Rechner! Dadurch können Sie Hudson ganz allgemein zur zentralen Protokollierung von automatisiert ausgeführten Prozessen einsetzen.

"Free Style"-Softwareprojekt bauen

Dieses Profil ist das meistgenutzte in Hudson. Hudson baut Ihr Projekt, wobei Sie universell jedes SCM System mit jedem Build-Verfahren kombinieren können. Dieses Profil ist nicht nur auf das Bauen von Software beschränkt, sondern kann darüber hinaus auch für weitere Anwendungsgebiete verwendet werden.

Multikonfigurationsprojekt bauen (alpha)

Dieses Profil eignet sich sehr gut für Projekte mit zahlreichen Konfigurationen, die etwa in unterschiedlichen Umgebungen getestet oder plattformspezifisch gebaut werden müssen.

Maven 2 Projekt bauen

Dieses Profil baut ein Maven 2 Projekt. Hudson wertet dabei Ihre POM Dateien aus und reduziert damit den Konfigurationsaufwand ganz erheblich. Zwar befindet sich dieses Profil zur Zeit noch in der Entstehungsphase, es ist aber bereits einsetzbar, um Rückmeldungen von Anwendern zu sammeln.

Kopiere bestehenden Job

Damit kann ein anderer Job kopiert werden.

Zur Umsetzung der Projekte in Hudson wurden ausschließlich das „Free Style“-Profil verwendet, da es sich flexibler konfigurieren lässt

4.2.2.1.2 Verwendete Plugins

Bei der Umsetzung der Projekte spielten außerdem viele Plugins eine Rolle, die den Funktionsumfang von Hudson erweitern. Die Installation und Verwaltung der Plugins wird im Kapitel 61 genau beschrieben. Welche Plugins in den einzelnen Projekten zum Einsatz kamen und wie sie konfiguriert wurden, wird in der Umsetzung erläutert.

4.2.2.1.3 Konfiguration von Hudson

Bei der Umsetzung der Projekte wurden zur Durchführung der Build-Schritte verschiedene Kommandozeilen-Programme verwendet. In den Verwaltungseinstellungen von Hudson wurden mehrere globale Variablen definiert, um auf diese Programme zu verweisen. Durch diese zentrale Verwaltung der Programm-Pfade verringerte sich der Konfigurationsaufwand.

4.2.2 Allgemeines zur Umsetzung in Hudson

Der folgende Ausschnitt zeigt die angelegten Variablen:

Name	DEVENV_VS2008
Wert	"C:\Program Files\Microsoft Visual Studio 9.0\Common7\IDE\devenv.com"
Name	NUNIT_2_CONSOLE
Wert	C:\tools\NUnit-2.5.2.9222\bin\net-2.0\nunit-console.exe
Name	ZIP7
Wert	C:\tools\7za465\7za.exe

Abbildung 34: Hudson - Konfiguration - Variablen

Die Variable `DEVENV_VS2008` verweist auf das Kommandozeilen-Programm von MS Visual Studio, `NUNIT_2_CONSOLE` verweist auf das Kommandozeilen-Programm des Testframeworks NUnit und `ZIP7` verweist auf das Archivierungs-Programm 7-Zip⁴⁴.

4.2.3 C#-Projekt

In diesem Projekt stellt eWorks für einen Kunden ein Client-Programm bereit. Es handelt sich bei dem Client-Programm um eine auf dem .NET-Framework basierende Desktop-Anwendung. Dieses Projekt hat kein genaues zeitliches Ende, da es sich um ein Support-Projekt handelt, bei dem immer mal wieder Arbeit anfällt. So soll der CIS in diesem Projekt nicht nur bis zur Fertigstellung des Produkts eingesetzt, sondern auch in den darüber hinaus gehenden Support-Arbeiten zur Qualitätssicherung benutzt werden.

4.2.3.1 Ausgangslage

In dem Projekt arbeiteten 2 Entwickler, welche für die Bearbeitung des Quellcode die grafische Entwicklungsumgebung MS Visual Studio 2008 verwendeten. Der Build-Vorgang wurde ebenfalls mit diese Entwicklungsumgebung durchgeführt und keine Build-Tools zum bauen verwendet.

Das Testen der Anwendung war ebenfalls ein manueller Prozess, der von den Entwicklern mit Hilfe eines schriftlichen Testprotokolls durchgeführt wurde. Für dieses Projekt wurden bisher noch keine UnitTests erstellt.

Die Erstellung neuer Releases der Anwendung bestand zum Großteil aus manuellen

⁴⁴ <http://www.7-zip.org/>

Arbeitsschritten. Dazu zählt das Auschecken des Quellcodes aus der Versionsverwaltung, bauen der Anwendung mit Release-Einstellungen, das Testen der Anwendung und das Erstellen des Paketes zur Weitergabe an den Kunden. Die Paket-Erstellung wurde mit Hilfe eines Windows Shell-Skriptes automatisiert. Als Eingabeparameter erwartete das Skript die Versionsnummer der Anwendung. Diese Nummer befand sich in einer Konfigurationsdatei des Projekts und wurde vor einem Release von einem der Entwickler angepasst. Das erstellte Release wurde darauf hin als ZIP-Archiv komprimiert und den Testern oder Kunden zur Verfügung gestellt.

Manche von der Anwendung verwendete Bibliotheken waren außerdem verschlüsselt und erwarteten bei der Kompilierung die Eingabe eines gültigen Passworts.

Wie bereits angemerkt, handelte es sich bei diesem Projekt um eine Support-Arbeit. Schnelles Reagieren und das zeitnahe Bereitstellen von neuen Releases war immens wichtig. Der Einsatz von Hudson soll in diesen Projekt vor allem die Bereitstellung der fertigen ZIP-Releases vereinfachen und dadurch sehr viel Arbeitsaufwand einsparen.

4.2.3.2 Umsetzung in Hudson

Für dieses Projekt wurde ein „Free Style“-Job angelegt und konfiguriert.

Allgemeine Konfiguration:

Damit nicht jeder Mitarbeiter die Konfiguration verändern kann, wurde die Projektbasierte Sicherheit aktiviert und die Zugriffsrechte für die Entwickler des Projekts eingestellt. Dabei wurden dem Projektleiter zusätzlich die Rechte zum Ändern und Löschen der Konfiguration zugeteilt:


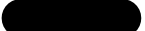

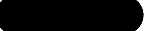
Benutzergruppe	Job					Starten	
	Delete	Configure	Read	Build	Workspace	Delete	Update
 	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
 	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
Anonymous	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Abbildung 35: C#-Projekt - Konfiguration - Benutzerrechte

Wie bereits erwähnt, benutzten die Entwickler ein Skript für die Zusammenstellung von Weitergabe-Paketen an den Kunden. Dieses Skript erwartete als Parameter die Release-Nummer und sollte auch in dem Job weiterverwendet werden. In Hudson lassen sich für die Jobs Parameter definieren, die beim manuellen Starten des Jobs übergeben werden

4.2.3.2 Umsetzung in Hudson

Text-Parameter

Name

Vorgabewert

Beschreibung

Abbildung 36: C#-Projekt - Konfiguration - Parameter

können. Wird der Job automatisch (z.B. durch eine Änderung des Quellcodes in der Versionsverwaltung) ausgelöst, werden die definierten Standardwerte für die Parameter verwendet. Damit die Entwickler bei Bedarf eine eigene Nummer übergeben können, wurde ein Text-Parameter hinzugefügt (Abb. 36).

In dem Bereich „Source Code Management“ wurde Subversion als Versionsverwaltung eingestellt und die notwendigen Daten wie die Repository URL und die Zugangsdaten übergeben. Außerdem wurde die Option „Update-Kommando verwenden“ aktiviert. Dadurch wird der ausgecheckte Quellcode mit dem Update-Kommando aktualisiert, statt ihn jedes mal zu Löschen und neu auszuchecken. Dies spart viel Zeit.

In dem Bereich „Build Auslöser“ wurde der Job so eingestellt, dass Hudson alle 5 Minuten die Versionsverwaltung abfragt und bei einer Änderung den Job auslöst. Der Zeitplan wird dabei mit Hilfe einer Cron⁴⁵-Syntax definiert, welche in der integrierten Hilfe zum dem Eingabefeld ausführlich beschrieben wird. Für eine Wiederholung alle 5 Minuten verwendet man den Ausdruck „*/5 * * * *“. Da in Subversion jede Änderung des Quellcodes eine neue Revisionsnummer erzeugt, ist die Prüfung auf Änderungen recht einfach und ressourcensparend. Hudson vergleicht dazu einfach die aktuelle Revisionsnummer mit der des letzten Builds.

Um zu vermeiden, dass ein „hängender“ Job die Ausführung weiterer Jobs blockieren kann, wurde in dem Bereich „Build Umgebung“ ein zusätzliches Plugin namens „Build-timeout“ (Beschreibung in 46) eingesetzt. Damit wurde der Job so eingestellt, dass er automatisch nach 10 Minuten abbricht und den Build als fehlgeschlagen kennzeichnet:

⁴⁵ <http://en.wikipedia.org/wiki/Cron>

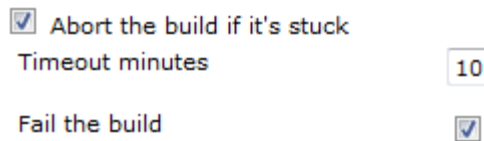


Abbildung 37: C#-Projekt - Konfiguration - Timeout

Build-Verfahren:

In dem Bereich „Build Verfahren“ wurden die einzelnen Build-Schritte mit Hilfe von Windows Shell-Skripten und dem Plugin „MSBuild“ umgesetzt. Die Entwickler setzten zur Zusammenstellung von Übergabe-Paketen an den Kunden bereits ein Windows Shell-Skript ein und kannten sich mit der Skript-Sprache aus. So lag es nahe auch für die Umsetzung der Build-Schritte Windows Shell-Skripte zu verwenden. Mit dem Plugin „MSBuild“ war es möglich das Build-Tool MSBuild zum Kompilieren der MS Visual Studio Projekte zu benutzen.

In dem **ersten Schritt** wurde das Arbeitsverzeichnis des Jobs aufgeräumt und alte Build-Artefakte gelöscht:

```
[1] @echo off & setlocal
[2] SET DEL_PATTERN=*-Client-*.zip
[3]
[4] echo.
[5] echo --- Lösche alte Zips (%DEL_PATTERN%) ---
[6] del %WORKSPACE%\%DEL_PATTERN%
[7] echo.
```

In dem **zweiten Schritt** wurde die Anwendung mit Hilfe des Plugins „MSBuild“ unter Verwendung der „Release“-Konfiguration neu erstellt. Die Option TargetFrameworkVersion war in diesem Fall wichtig, da die Anwendung auf .Net der Version 2.0 ausgelegt wurde:

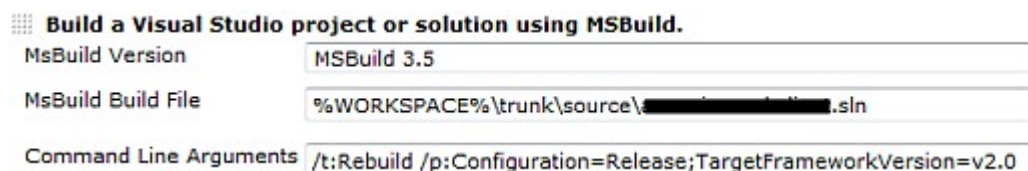


Abbildung 38: C#-Projekt - Konfiguration - MSBuild

Als **dritten Schritt** wurde wieder ein Shell-Skript hinzugefügt. Dieses erfüllte nacheinander verschiedene Aufgaben. Das vollständige Skript befindet sich im Anhang

4.2.3.2 Umsetzung in Hudson

und wird hier nur Ausschnitten abgebildet. Zuerst wurde geprüft, ob der im Job definierte Parameter *RELEASE_VERSION* übergeben wurde (Zeile 2). War das nicht der Fall, wurde die Versionsnummer mit Hilfe des Windows Kommandozeilen-Programms *findstr* aus einer definierten Datei *ASSEMBLY_INFO* ausgelesen (Zeile 4):

```
[1] IF "%RELEASE_VERSION%" == "" (
[2]   SET VPATTERN="^\[assembly: AssemblyVersion(\\".*\\")\]"
[3]   echo Lese Datei %ASSEMBLY_INFO% aus
[4]   FOR /f "tokens=2 delims=()" %%i IN ('findstr /R /C:%VPATTERN
% %ASSEMBLY_INFO%') DO SET RELEASE_VERSION=%%~i
[5] )
```

Im Anschluss wurde die Versionsnummer an das bereits erwähnte Skript zur Paket-Erstellung übergeben:

```
[1] call release.bat %RELEASE_VERSION% 2>&1
```

Das Skript *release.bat* legte für das Paket ein Verzeichnis an, welches im Anschluss für die Weitergabe an den Kunden als ZIP-Archiv komprimiert wurde:

```
[1] :: übernehme Variablen aus der release.bat
[2] IF "%filename%" == "" (
[3]   call :error "Dateiname für die zip ist leer"
[4] )
[5] SET ZIP_NAME="%filename%.zip"
[6] IF "%fullDestDir%" == "" (
[7]   call :error "Release-Verzeichnis zum zipen nicht gefunden"
[8] )
[9] SET SOURCE_FILES="%fullDestDir%*"
[10]
[11] echo Erstelle Zip Datei %ZIP_NAME%
[12] echo.
[13] %ZIP7% a %WORKSPACE%\%ZIP_NAME% -tzip %SOURCE_FILES%
```

Zum Anlegen des ZIP-Archivs wurde das Kommandozeilen-Programm *7-Zip* verwendet. Die Variable *ZIP7* (Zeile 13) wurde in den Verwaltungseinstellungen von Hudson definiert und verweist auf das Programm.

Nach dem Erstellen des ZIP-Archivs wurde das von *release.bat* erzeugte Verzeichnis gelöscht.

Post-Build:

Der Job wurde so konfiguriert, dass er nach dem Build weitere Aufgaben erfüllt. Zuerst wurde die erzeugte ZIP-Datei als Artefakt des Builds archiviert:

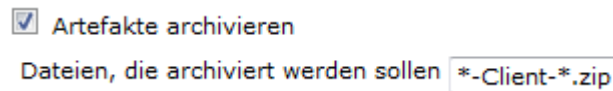


Abbildung 39: C#-Projekt - Konfiguration - Artefakt

Diese Artefakte speichert Hudson zusammen mit den Ergebnissen des Builds ab. Über die Weboberfläche können die Artefakte jederzeit heruntergeladen und an den Kunden weitergegeben werden.

War der Build erfolgreich, wurde mit dem Plugin „Task-Scanner“ (Vorstellung in Kapitel 50) alle C# Dateien (Dateiendung „.cs“) nach offenen Punkten durchsucht. Dabei wurden für die Kennzeichnung der offenen Punkte und dessen Prioritäten folgende Texte benutzt:

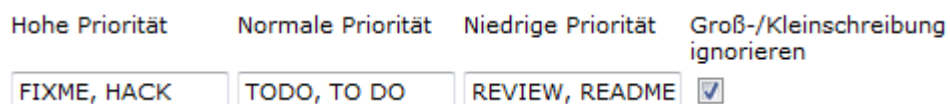


Abbildung 40: C#-Projekt - Konfiguration - Offene Punkte

Unabhängig vom Status des Builds wurden mit Hilfe des Plugins „Warnings“ die Konsolenausgabe nach Compiler-Warnungen des Build-Tools MSBuild durchsucht.

Das Plugin „Continuous Integration Game“ wurde für diesen Job aktiviert, damit die Entwickler Punkte für das Spiel sammeln können.

Bei einem fehlgeschlagenen Build werden die Entwickler und Projektleiter davon unterrichtet. Dazu wurden in der Einstellung „E-Mail-Benachrichtigung“ die E-Mail-Adressen angegeben.

4.2.4 VB.NET-Projekt

Das nächste Projekt basiert ebenfalls auf dem .NET-Framework und wird von eWorks in Form zweier Desktop-Anwendungen und einer Web-Anwendung bereitgestellt. Bei diesem Projekt handelt es sich um eine Projektarbeit, die über einen mehrere Monate andauernden Zeitrahmen hinaus verläuft und in dem bis zu 4 Software-Entwickler zeitgleich arbeiten. Darunter befand sich auch ein externer Mitarbeiter, der nicht in den Büros von eWorks arbeitete.

4.2.4.1 Ausgangslage

In diesem Projekt setzten die Entwickler ebenfalls die grafische Entwicklungsumgebung MS Visual Studio zum bauen der Anwendung ein. Weitere Build-Tools wurden nicht verwendet.

Zur frühzeitigen Aufdeckung von Programmierfehlern setzte eWorks in diesem Projekt UnitTests ein. Diese Tests wurden von den Entwicklern manuell zum Testen der Änderungen vor der Übernahme in die Versionsverwaltung ausgeführt.

Auch in diesem Projekt war die Erstellung von Releases ein manueller Prozess. Für die Desktop-Anwendungen wurden mit der Entwicklungsumgebung MS Visual Studio jeweils eine Windows Setup-Dateien generiert und den Testern und Kunden bereitgestellt. Die Web-Anwendung wurde manuell mit Hilfe der Entwicklungsumgebung MS Visual Studio veröffentlicht und anschließend von Hand auf einem Test-Server installiert. Diese Installation der Web-Anwendung stand dann dem Kunden zu Demonstrationszwecken und Tests zur Verfügung.

Neben dem Kompilieren des Quellcodes gesellte sich hier auch der Aspekt der Software-Verteilung hinzu, da in diesem Projekt das Release nicht nur bereitgestellt, sondern im Falle der Web-Anwendung auch auf einem anderen Rechner installiert wurde.

4.2.4.2 Umsetzung in Hudson

Für dieses Projekt wurden drei „Free Style“-Jobs angelegt und konfiguriert. In dem Haupt-Job wurde die Anwendung gebaut und getestet. Im Anschluss wurde von einem zweiten Job die Webanwendung zusammengestellt und auf den Test-Server kopiert. Der dritte Job wurde auf dem Test-Server ausgeführt und installierte bei Bedarf die kopierte Webanwendung auf dem Test-Server.

4.2.4.2.1 Haupt-Job

Allgemeine Konfiguration:

In diesem Projekt wurde die Sicherheit aktiviert und die Benutzerrechte an die Entwickler verteilt. Der Projektleiter hatte dabei zusätzlich das Recht die Konfiguration









Benutzergruppe	Job					Starten	
	Delete	Configure	Read	Build	Workspace	Delete	Update
 	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
 	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
 	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
 	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
Anonymous	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Abbildung 41: VB.NET Projekt – Haupt-Job Konfiguration - Benutzerrechte

des Jobs zu Ändern:

Auch in diesem Projekt wurde ein Parameter definiert. Dabei wurde der Parameter von Typ Auswahl verwendet, mit dem eine Liste möglichen Optionen festgelegt werden kann. Mit dem angelegten Parameter *ARCHIV* konnte man beim manuellen Starten des Jobs auswählen, ob dem Release eine weitere Installations-Datei hinzugefügt werden soll (Abb. 42). Die oberste Auswahlmöglichkeit war dabei der Standardwert, der beim automatischen Start des Jobs durch einen Build-Auslöser verwendet wurde.

Auswahl

Name

Auswahlmöglichkeiten

Beschreibung

Abbildung 42: VB.NET-Projekt – Haupt-Job Konfiguration - Parameter

Als Versionsverwaltung wurde auch in diesem Projekt Subversion ausgewählt und die entsprechenden Zugangsdaten eingegeben. Um das Aktualisieren des ausgecheckten Quellcodes zu beschleunigen wurde ebenfalls die Option „Update-Kommando verwenden“ aktiviert.

Der Build Auslöser wurde so eingestellt, dass er die Versionsverwaltung alle 5 Minuten nach Änderungen abfragt. Der entsprechende Eintrag nach Cron-Syntax für den Zeitplan lautet „*/5 * * * *“.

Auch in diesem Projekt wurde das Plugin „Build-timeout“ verwendet, um den Job automatisch nach 10 Minuten abzurechnen und als fehlgeschlagen zu markieren.

Build-Verfahren:

Im Build-Verfahren wurden Windows Shell-Skripte und das Plugin „MSBuild“

4.2.4.2 Umsetzung in Hudson

verwendet. Die vollständigen Skripte befinden sich im Anhang (Kapitel 124).

In dem **ersten Schritt** wurde das Arbeitsverzeichnis des Jobs aufgeräumt und dabei alte Artefakte und Berichte des Testframeworks NUnit gelöscht.

In dem **zweiten Schritt** wurde die Anwendung mit dem Build-Tool MSBuild unter Verwendung der „Release“-Konfiguration neu erstellt:

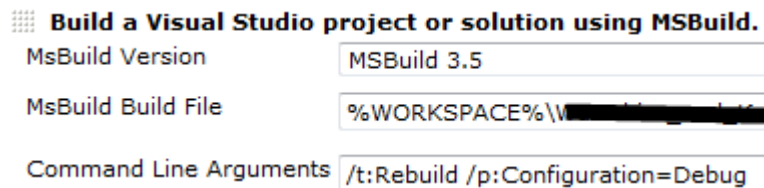


Abbildung 43: VB.NET-Projekt – Haupt-Job Konfiguration - MSBuild

Der **dritte Schritt** führte die UnitTests des Projekts aus. Dabei wurde ein Shell-Skript zum Aufruf des Testframeworks NUnit verwendet:

```
[1] %NUNIT_2_CONSOLE% %WORKSPACE%
%\KUNDENNAME\PROJEKT\source\ABC\ABC.Unittests\bin\Debug\ABC.Unitt
ests.dll /xml=NUnitResults.xml
```

Danach folgte die Erstellung der Setup-Dateien für die zwei Desktop-Anwendungen. Die Entwickler verwendeten dazu die Visual Basic Setup Projekte aus MS Visual Studio. Das Problem bei diesen Setup-Projekten ist, dass sie von dem Build-Tool MSBuild nicht unterstützt werden und mit dem Kommandozeilen-Programm devenv.com von MS Visual Studio gebaut werden müssen. Deshalb wurde für den **vierten Schritt** ein weiteres kleines Shell-Skript angelegt. Die folgende Codezeile zeigt den genauen Befehl zum bauen einer der Desktop-Anwendungen:

```
[1] %DEVENV_VS2008% %WORKSPACE%
%\KUNDENNAME\PROJEKT\source\ABC\ABC.sln /rebuild Debug /project
"ABC-Manager Setup" /projectconfig Debug
```

Die Variable *DEVENV_VS2008* verweist dabei auf das Programm devenv.com.

Der **fünfte Schritt** erstellte aus den Desktop-Anwendungen ZIP-Dateien für die Archivierung. Je nach Wert des Job-Parameters *ARCHIV* enthielten die ZIP-Archive eine zusätzliche Installationsdatei.

Post-Build:

In diesem Job wurden die erstellten ZIP-Dateien als Artefakte des Builds gesichert. Dabei wurde der Parameter *ARCHIV* in dem Eingabefeld für die Dateiangabe

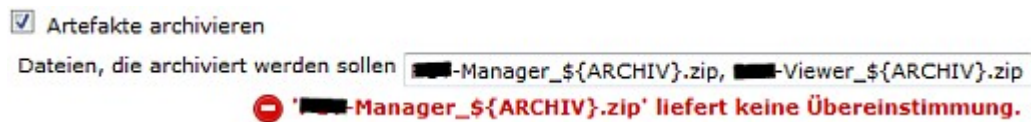


Abbildung 44: VB.NET-Projekt – Haupt-Job Konfiguration - Artefakte

verwendet. Da der Wert des Parameters erst zur Laufzeit des Jobs gesetzt wird, kam es in der Weboberfläche zu einer Fehlermeldung. Diese könnte jedoch ignoriert werden.

Dieser Job wurde durch die Option „Weitere Projekte bauen“ so eingestellt, dass es automatisch den Web-Job auslöst. Dazu musste nur der Projektname des Web-Jobs angegeben werden.

Zu den Post-Build Aktionen gehörte auch das Auslesen von Testberichten. In dem Fall wurde mit Hilfe des Plugins „NUnit“ der Testbericht von NUnit ausgelesen. Danach stehen die Testergebnisse in Hudsons Weboberfläche in Form von Statistiken und Grafiken zur Verfügung.

Genauso wie im C#-Projekt wurden in diesem Job ebenfalls die Plugins „Task-Scanner“ „und Warnings“ verwendet um die offene Punkte bzw. Compiler-Warnungen zu sammeln. Die Plugins wurden im Kapitel 50 bereits genauer vorgestellt.

Da es sich in diesem Projekt um eine VB.NET Anwendung handelte wurden für die offenen Punkte jedoch alle Visual Basic Dateien (Dateiendung „.vb“) durchsucht. Zur Kennzeichnung der Prioritäten wurden die gleichen Kennzeichen benutzt:



Abbildung 45: VB.NET-Projekt – Haupt-Job Konfiguration - offene Punkte

Da auch hier das Build-Tool MSBuild zum Einsatz kam, wurde das Plugin Warnings so eingestellt, dass es die Compiler-Warnungen von MSBuild sammelt.

Auch in diesem Projekt wurde das Continuous Integration Spiel aktiviert und die Entwickler konnten durch die erfolgreichen Builds Punkte sammeln.

Bei einem fehlerhaften Build wurde eine Nachricht an die festgelegten E-Mail-Adressen der Entwickler und Projektleiter versendet. Dazu wurden in der „E-Mail-Benachrichtigung“ alle E-Mail-Adressen (getrennt durch Leerzeichen) angegeben.

4.2.4.2.2 Web-Job

Allgemeine Konfiguration:

In diesem Job müssen keine Benutzerrechte für die Entwickler eingestellt werden, da der Job durch den Haupt-Job ausgelöst werden soll.

In dem Bereich „Source Code Management“ wurde keine Versionsverwaltung eingestellt. Dieser Job sollte das Arbeitsverzeichnis (und damit den Quellcode) des Haupt-Jobs verwenden und selber keinen Quellcode auschecken. Dies wurde durch die Option „Verzeichnis des Arbeitsbereichs anpassen“ in dem Bereich „Erweiterte Projekteinstellungen“ erreicht. Dort ließ sich der Pfad zum Arbeitsverzeichnis des Haupt-Jobs angeben. Um sicherzustellen, dass der Web-Job nicht ausgeführt wurde solange ein Haupt-Job lief, wurde zusätzlich noch die Option „Build blockieren, solange vorgelagertes Projekt gebaut wird.“ aktiviert.

In dem Bereich „Build Auslöser“ wurde bereits automatisch der Projektname des Haupt-Jobs als Auslöser des Jobs eingetragen.

Als Timeout Wert für das Abbrechen des Jobs wurden 5 Minuten eingestellt, da der Job normalerweise nicht länger als eine Minute dauerte.

Build-Verfahren:

In diesem Job wurden ebenfalls Windows Shell-Skripte und das Build-Tool MSBuild benutzt. Die vollständigen Shell-Skripte befinden sich im Anhang (Kapitel 126).

Als **ersten Schritt** wurden hier mit Hilfe eines Shell-Skriptes alte Artefakte gelöscht.

Danach folgte im **zweiten Schritt** die Veröffentlichung der Webanwendung mit Hilfe von MSBuild:

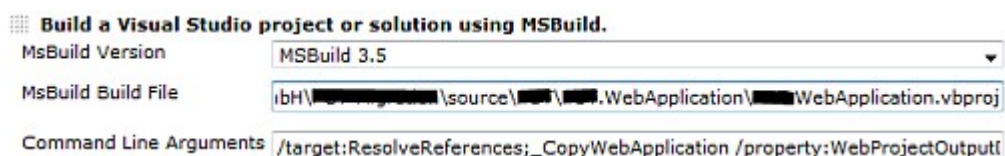


Abbildung 46: VB.NET-Projekt - Web-Job Konfiguration - MSBuild

Die Veröffentlichung einer Webanwendung mit einem Kommandozeilen-Befehl durchzuführen war nicht so einfach und erforderte eine Reihe zusätzlicher Argumente, die in dem Feld „Command Line Arguments“ eingetragen werden mussten:

```
[1] /target:ResolveReferences;_CopyWebApplication
/property:WebProjectOutputDir=%WORKSPACE%\WebPublish/;OutDir=
%WORKSPACE%\WebPublish\bin/
```

In dem **dritten Schritt** wurde aus der veröffentlichten Webanwendung und verschiedene Konfigurationsdateien ein ZIP-Archiv erstellt und im Anschluss auf den Test-Server kopiert.

Post-Build:

Die erstellte ZIP-Datei wurde als Artefakt des Builds gesichert.

Da in dem Build-Verfahren das Build-Tool MSBuild zum Einsatz kam, wurde auch in diesem Job mit Hilfe des Plugins „Warnings“ die Konsolenausgabe nach Compiler-Warnungen durchsucht.

Bei einem fehlerhaften Build sendete der Job eine E-Mail-Benachrichtigung an alle betroffenen Entwickler.

4.2.4.2.3 Test-Server-Job

Allgemeine Konfiguration:

Dieser Job wurde mit Hilfe eines Agenten auf dem Test-Server ausgeführt. Dazu musste das Projekt in der Konfiguration an den Knoten des Test-Servers gebunden werden:

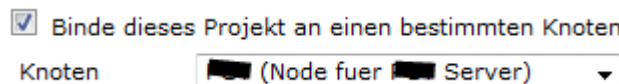


Abbildung 47: VB.NET-Projekt - Test-Server-Job Konfiguration - Knoten

Die Projekt-basierte Sicherheit wurde aktiviert und den Entwicklern des Projekts die Benutzerrechte zugewiesen:

Benutzergruppe	Job					Starten	
	Delete	Configure	Read	Build	Workspace	Delete	Update
[Redacted]	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
[Redacted]	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
[Redacted]	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
[Redacted]	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Anonymous	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Abbildung 48: VB.NET-Projekt - Test-Server-Job Konfiguration - Benutzerrechte

Mit Hilfe eines Parameters vom Typ Datei konnte man ZIP-Datei mit der zu

4.2.4.2 Umsetzung in Hudson

installierende Webanwendung übergeben. Normalerweise wurde durch dem Job die aktuellste Webanwendung aus dem Web-Job installiert, aber mit dem Parameter war es möglich eine bestimmte Version der Webanwendung zu installieren:

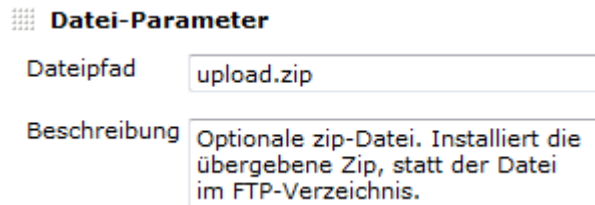


Abbildung 49: VB.NET-Projekt - Test-Server-Job Konfiguration - Parameter

Die übergebene Datei wurde in dem Arbeitsverzeichnis des Agenten unter dem Namen *upload.zip* gespeichert.

Für den Job wurde keine Versionsverwaltung verwendet.

Der Job wurde so eingestellt, dass er nach spätestens 5 Minuten abbricht.

Build-Verfahren:

Das Build-Verfahren bestand nur aus 2 Shell-Skripten.

Das erste entpackte die ZIP-Datei der Webanwendung in das Projektverzeichnis des Web-Servers. Normalerweise wurde dazu die ZIP-Datei aus dem Web-Job verwendet, es sei denn es wurde eine Datei mit dem Parameter des Jobs übergeben. Damit die Konfigurationsdatei der bereits installierten Webanwendung nicht verloren ging, wurde die Datei *web.config* vor dem Entpacken gesichert und im Anschluss wieder hergestellt.

Für den Fall, dass die Webanwendung über den Datei-Parameter übergeben wurde, Löschte ein zweites Shell-Skript die Datei *upload.zip*.

Post-Build:

Bei einem fehlerhaften Build wurden die Entwickler durch eine E-Mail darüber unterrichtet.

4.2.5 PHP-Projekt

In diesem Projekt wird für einen Kunden ein Online-Shop entwickelt. Die Umsetzung erfolgt in der Programmiersprache PHP und mit Hilfe der Standardsoftware OXID

eShop⁴⁶. Es handelt sich dabei um ein sehr neues Projekt, dessen Entwicklung erst im Dezember 2009 begonnen wurde.

4.2.5.1 Ausgangslage

Die Entwickler verwendeten die Entwicklungsumgebung Zend Studio⁴⁷ zur Bearbeitung des Quellcodes und für die Entwicklung von UnitTests wurde das Testframework PHPUnit⁴⁸ eingesetzt.

Zur Durchführung von Integrations-Tests und zur späteren Demonstration für den Kunden wurde ein virtueller Rechner eingerichtet, auf der eine lauffähige Version der Anwendung installiert werden kann. Dazu wurde von den Entwicklern auf der Rechner ein Web-Server eingerichtet und für das Projekt konfiguriert. Zur Installation einer neuen Version musste der Quellcode nur noch auf den Test-Server kopiert werden.

4.2.5.2 Umsetzung in Hudson

Dieses Projekt wurde mit Hilfe eines „Free Style“-Jobs umgesetzt.

Allgemeine Konfiguration:

Wie in den andern Projekten auch, wurde in diesem Job die Projekt-basierte Sicherheit aktiviert und den Entwicklern die Benutzerrechte zugewiesen:


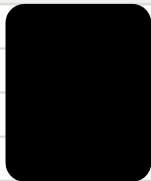



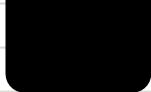

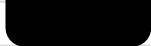
Benutzergruppe	Job					Starten	
	Delete	Configure	Read	Build	Workspace	Delete	Update
 	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
 	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
 	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
 	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
Anonymous	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Abbildung 50: PHP-Projekt - Konfiguration - Benutzerrechte

Als Versionsverwaltung wurde auch hier wieder Subversion eingestellt und die notwendigen Angaben wie Repository-URL und Benutzerdaten gemacht.

Als Build-Auslöser für diesen Jobs wurde eine Änderung des Quellcodes in der Versionsverwaltung konfiguriert. Dabei fragte Hudson die Versionsverwaltung alle 5 Minuten nach einer Änderung ab. Der Zeitplan musste dazu mit dem Wert „*/5 * * * *“

46 <http://shoptimax.de/ebusiness/oxid-eshop/>

47 <http://www.zend.com/de/products/studio/>

48 <http://www.phpunit.de/>

4.2.5.2 Umsetzung in Hudson

eingestellt werden.

Build-Verfahren:

In diesem Projekt wurde ein Phing Build-Skript für die Definition des Build-Verfahrens verwendet. Das vollständige Build-Skript befindet sich im Anhang (Kapitel 128).

Der Aufruf geschah mit Hilfe eines Kommandozeilenbefehls:

```
[1] phing -f %WORKSPACE%\Web-Shop\build.xml build deploy-test  
-Dws=%WORKSPACE%
```

Dadurch wurde das Build-Tool Phing angewiesen das angegebene Build-Skript zu starten und die darin enthaltenen Targets „build“ und „deploy-test“ auszuführen. Der Parameter *ws* übergibt dabei den Pfad des Arbeitsverzeichnisses des Jobs an das Build-Skript.

Durch das Build-Target „build“ wurden die Targets „clean“, „prepare“ und „phpunit“ aufgerufen:

```
[1] <target name="build"  
[2]   depends="clean, prepare, phpunit"/>
```

Das Target „clean“ Löschte das alte Build-Verzeichnis.

Das Target „prepare“ bereitete den Build durch das Anlegen eines Build-Verzeichnisses vor. Darin wurden unter anderem auch die Berichte des verwendeten Testframeworks PHPUnit abgelegt.

Durch das Target „phpunit“ erfolgte die Ausführung der UnitTests.

```
[1] <target name="phpunit">  
[2]   <echo msg="PHP Unit..." />  
[3]   <exec command="phpunit  
--log-junit ${builddir}/logs/phpunit.xml  
--log-pmd ${builddir}/logs/phpunit.pmd.xml  
--coverage-clover ${builddir}/logs/coverage/clover.xml  
--coverage-html ${builddir}/logs/coverage/  
${testsdire}"/>  
[4] </target>
```

Das interessante an PHPUnit ist, dass es in der Lage ist, seine Testberichte in anderen Formaten auszugeben. In dem Fall wurde durch den Parameter „--log-junit“ das Format eingestellt, dass vom Testframework JUnit verwendet wird. Da Hudson JUnit direkt unterstützt, ist es möglich die Testberichte mit dem Format auszulesen.

Das Testframework PHPUnit ist auch in der Lage einen Bericht über die Codeabdeckung der UnitTests zu erstellen. Dies geschah mit dem Parameter:

```
„--coverage-clover ${builddir}/logs/coverage/clover.xml“
```

Dabei wurde der Bericht in dem Format abgespeichert, welches von dem Metrik-Tool Clover verwendet wird. Das Auslesen von Clover-Berichten wurde durch Plugins ebenfalls von Hudson unterstützt.

Durch das Target „deploy-test“ wurde die Anwendung auf den Test-Server kopiert:

```
[1] <target name="deploy-test">
[2]     <echo msg="Kopiere auf Testserver..." />
[3]     <!-- zugriff auf den testserver -->
[4]     <exec command="net use ${testserverdir} /user:USERNAME
PASSWORD" />
[5]     <!-- kopieren der dateien -->
[6]     <copy todir="${testserverdir}"
[7]         overwrite="false"
[8]         includeemptydirs="true" >
[9]         <fileset dir="${copyfilesdir}">
[10]             <include name="**/*" />
[11]         </fileset>
[12]     </copy>
[13]     <!-- Lösche alte temporäre Dateien auf dem Test-Server -->
[14]     <echo msg="Lösche Dateien im tmp Verzeichnis..." />
[15]     <delete>
[16]         <fileset dir="${testserverdir}\tmp">
[17]             <include name="**/*" />
[18]         </fileset>
[19]     </delete>
[20] </target>
```

In dem Build-Skript wurden auch viele weitere Targets zum Starten der Dokumentationserstellung oder Metrik-Tools angelegt. Jedoch wurden diese bisher noch nicht verwendet, da die Entwickler bisher noch keine eigenen Quellcode erstellt hatten. Es machte auch wenig Sinn die Metrik-Tools zur Qualitätssicherung der verwendeten Standardsoftware eShop einzusetzen, solange deren Quellcode-Dateien nicht von den Entwicklern verändert wurden.

4.2.5.2 Umsetzung in Hudson

Zur Generierung der Quellcode-Dokumentation wurde in dem Build-Skript ein Target zur Verwendung des Programms PHPDoc⁴⁹ erstellt.

Für die Metrik-Tools PHPCPD⁵⁰ (Erkennung von kopierten Code), PDepend⁵¹ (Prüfen der Code-Komplexität und der Abhängigkeiten zwischen den Klassen), PHPCS⁵² (Ratschläge zur Erstellung des Quellcodes) wurden ebenfalls schon Targets vorbereitet und müssten nurnoch in Hudson aufgerufen werden, sobald die Entwickler mit der Erstellung einer Quellcode-Dateien beginnen.

Post-Build:

In diesem Projekt war es geplant, die Entwicklung der Anwendung durch verschiedene Metrik-Tools zu unterstützen. Wie bereits erwähnt, machte dessen Einsatz jedoch zum gegenwärtigen Entwicklungsstand noch wenig Sinn, da man höchstens den Quellcode der verwendeten Standardsoftware eShop hätte prüfen können.

Die Berichte von PHPCPD könnten durch das Plugin „Violations“ als CPD-Bericht ausgelesen werden.

PHPCS ermöglichte die Erstellung von Berichten im Format des Metrik-Tools Checkstyle, welches ebenfalls von dem Plugin „Violations“ unterstützt wurde.

Der Aufruf von PDepend wurden so eingestellt, dass es Berichte im JDepend Format ausgab. Dieses Format könnte durch das Plugin JDepend ausgelesen werden.

Durch die Aktivierung der Option „Continuous Integration Game“ nahm dieses Projekt ebenfalls an dem Spiel teil.

Bei einem fehlerhaften Build wurde die Entwickler durch eine E-Mail benachrichtigt.

49 <http://www.phpdoc.org/>

50 <http://github.com/sebastianbergmann/phpcpd>

51 <http://pdepend.org/>

52 http://pear.php.net/PHP_CodeSniffer

5 Bewertung des CIS-Einsatzes

Der Continuous Integration Server Hudson wurde im Zuge der Diplomarbeit für 3 ausgewählte Projekte der Firma eWorks GmbH eingesetzt, um den Erfolg und die Notwendigkeit eines solchen Systems zu bewerten. Dadurch konnten viele Erfahrungen und Daten gesammelt werden, um den Einsatz von Hudson zu bewerten. Im Rahmen dieser Ausarbeitung finden nur die ersten beiden Projekte, das C#-Projekt und das VB.NET-Projekt eine Erwähnung, da die Ergebnisse des PHP-Projektes noch nicht schlussendlich bewertet werden können. Der Einsatz hat zu spät stattgefunden, um etwas über dessen Ergebnis berichten zu können.

Zu den messbaren Daten, die für eine Bewertung des CIS-Einsatzes herangezogen werden könnten, gehört die eingesparte Arbeitszeit bei der Erstellung neuer Releases. Die Mitarbeiter der Firma eWorks GmbH nutzen zur Erfassung ihrer Arbeitszeiten ein von ihnen entwickeltes Zeiterfassungssystem namens „TimeEngine⁵³“. Es handelt sich dabei um eine Webanwendung, die in der Lage ist zu den einzelnen Projekten und Aufgaben Berichte anzufertigen, die die Tätigkeiten und Zeitaufwände der Mitarbeiter aufführen. Daraus lassen sich die Aufwände zur Erstellung einzelner Releases zusammenfassen und die Zeitersparnis durch die Einführung des CIS berechnen.

Der Einsatz von Hudson lässt sich jedoch nicht nur aufgrund der eingesparten Arbeitszeit bewerten. Es spielen auch weitere Faktoren eine Rolle, die sich nicht so einfach in Zahlen ausdrücken lassen.

Die Darstellung aller Build-Ergebnisse in Hudsons Weboberfläche erleichtert es Nicht-Entwicklern einen Überblick über die Projekte zu gewinnen. Bei jedem Build können zusätzlich Informationen über den Zustand der Anwendung gesammelt werden. Diese Informationen werden in Hudson als Statistiken und Graphen dargestellt und sind jederzeit zugänglich. So ist es für jeden Mitarbeiter im Unternehmen möglich die Entwicklung der Projekte zu überwachen und bei Problemen gegenzusteuern.

Gleichzeitig wird in einem CIS auch das Wissen gesammelt, wie die Releases der einzelnen Projekte zu bauen sind. Mit dem Kunden vereinbarte Release-Übergaben werden nicht in Gefahr gebracht, wenn der dafür zuständige Entwickler aus diversen Gründen (Urlaub, auswärtiger Termin, Krankheit usw.) nicht zur Verfügung steht. Da

53 <http://www.timeengine.de/>

Hudson die Releases selbstständig baut, können sie von jedem beliebigen Mitarbeiter herausgegeben werden. Darüber hinaus sind in Hudson die Jobs für Projekte recht homogen. Die Konfiguration eines bestehenden Projektes kann somit herangezogen werden, um neue Projekte einzurichten. Dadurch ist auch das Einrichten von Projekten nicht zwingend von einer Person abhängig. In den eingesetzten Projekten hat sich vor allem der Punkt der unabhängigen Release-Herausgabe sehr bewährt, da es möglich war zu jedem Iterationsende ein Release zu erstellen. Eine Iteration besteht dabei aus einer Arbeitswoche und die Release-Herausgabe war zu jedem Freitag einer Woche angesetzt.

5.1 Projekte

Die bereits in Kapitel 4.2 vorgestellten Projekte unterscheiden sich sehr stark in ihrer Projektdauer und der Art der Anwendung. Deshalb wird der Einsatz von Hudson getrennt nach den Projekten beurteilt.

Beim Einsatz von Hudson wurden viele Probleme bereits vor dem Erstellen eines Releases erkannt. Dadurch wurde zusätzliche Zeit gespart, die man normalerweise zur Fehlersuche und Problemlösung benötigt hätte. Zu den häufigsten Problemen zählte das Fehlen neuer Quellcode-Dateien oder Bibliotheken, die von den Entwicklern versehentlich nicht in die Versionsverwaltung eingecheckt wurden. Ohne den CIS hätte dieses Problem unter Umständen sehr viel Zeit kosten können, z.B. wenn der Entwickler des neuen Codes aufgrund von Urlaub oder Krankheit nicht erreichbar gewesen wäre. Auch wurden bei den Projekten teilweise externe Mitarbeiter eingesetzt, deren Arbeitszeiten sich stark von denen der übrigen unterscheiden. Eine fehlende Datei hätte so bisweilen einen Ausfall von 12 oder mehr Stunden kosten können. Deshalb wurde bei den Berechnungen der Zeitersparnis in den einzelnen Projekten eine pauschale Zeit für Fehlersuche und Problemlösung hinzugefügt.

5.1.1 C#-Projekt

In diesem Projekt leisteten zwei Software-Entwickler einem Kunden Unterstützung für eine von eWorks weiterentwickelte Desktop-Anwendung und führten dabei auch Änderungen und Erweiterungen am Quellcode durch. Die Arbeiten in dem Projekt fand in einem Zeitraum von etwa einen Monat statt und der dafür eingeplante Zeitaufwand war mit 8 PT (Personen-Tagen) relativ gering. Der tatsächliche Aufwand betrug am

Ende des Projektzeitraumes 16 PT.

Vor dem Einsatz von Hudson wurden die neuen Releases der Anwendung weitgehend manuell erstellt. Das Paket für die Weitergabe an den Kunden wurde bereits mit einem Windows-Shell Skript automatisiert. Die folgende Tabelle listet die Vorgänge und den Zeitaufwand für die Erstellung eines Releases auf:

<u>Vorgang</u>	<u>Zeitaufwand (in Minuten)</u>
Subversion Checkout	1
Visual Studio starten und das Projekt öffnen	5
Kompilierung des Quellcodes	5
Weitergabe-Paket erstellen	10
ZIP-Archiv erstellen	2
Archiv auf den FTP-Server kopieren	5
<u>Summe:</u>	<u>28</u>

Tabelle 5.1 C# Projekt – Release-Zeitaufwand

Dem Zeitaufwand wird weiterhin ein pauschaler Wert von 15 Minuten für Fehlersuche und Problemlösung hinzugefügt. Dieser Wert ist relativ klein, da in dem Projekt nur wenige Fehler aufgetreten sind und keine externen Entwickler beteiligt waren.

Im Laufe des Projektzeitraumes wurden 7 Releases erstellt. Dadurch ergibt sich folgender Gesamtaufwand:

$$7 * (28 + 15) \text{ Minuten} = 301 \text{ Minuten} \approx 0,6 \text{ PT}$$

Text 6: C# Projekt - Release-Zeitaufwand

Durch den Einsatz von Hudson spart man in diesem Projekt also 43 Minuten Arbeitszeit pro Release. Insgesamt konnten in dem Projekt 0,6 PT eingespart werden, was immerhin 3,75% des gesamten Projektaufwandes ausmacht.

Seit der Einrichtung des Projekts wurden in Hudson 58 Builds ausgelöst, wobei 28 davon fehlschlugen. Der folgende Graph zeigt die Dauer der einzelnen Builds an, wobei die erfolgreichen blau und die fehlgeschlagen rot gekennzeichnet sind:

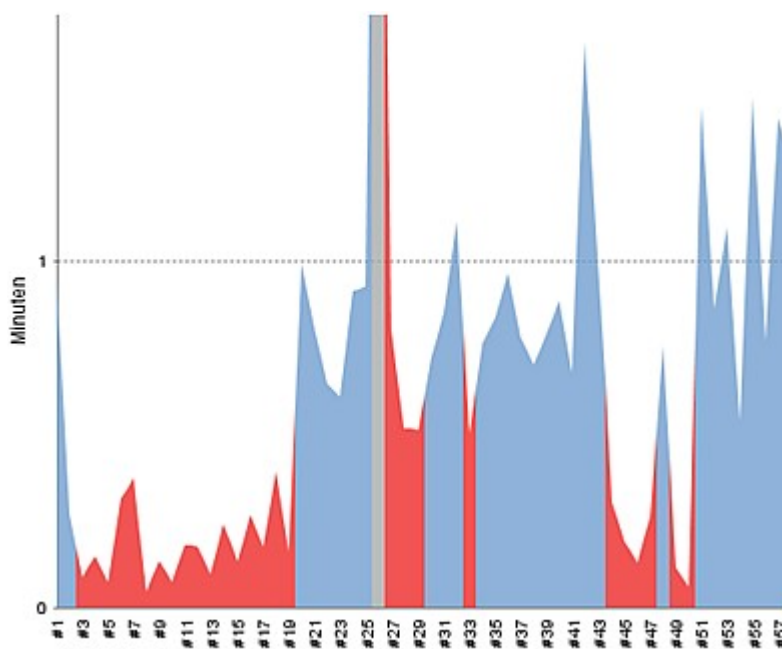


Abbildung 51: C# Projekt - Build-Dauer

Die ersten Fehlschläge (Build 3 bis 9) entstanden bei der Einrichtung des Jobs in Hudson aufgrund fehlender Entwickler-Werkzeuge und Einstellungen auf dem Rechner. Die Builds 10 bis 14 schlugen fehl, weil in dem Projekt verschlüsselte Bibliotheken verwendet wurden und der Compiler die Eingabe eines Authentifizierungsschlüssels erwartete. Dieses Problem trat bei Build 44-47 erneut auf. In den Builds 15-19 kam es ebenfalls zu Problemen, da sich in der Solution-Datei des Projekts fehlerhafte GUID-Referenzen auf bestimmte Unterprojekte befanden. Dadurch konnte der Compiler MSBuild die Unterprojekte und darin definierte Namensräume nicht finden. Verwendet man zur Kompilierung die Entwicklungsumgebung Visual Studio, tritt dieses Problem nicht auf, da es zur Referenzierung die Unterprojekt-Namen und nicht dessen GUIDs benutzt. Die Builds 26-29 und 33 schlugen aufgrund von Fehlern bei der Erweiterung der Build-Schritte in der Job-Konfiguration fehl. Da bei eWorks neue Server eingerichtet und der Netzwerk-Bereich der Server umgestellt wurde, konnte Hudson eine Zeit lang die Versionsverwaltung nicht erreichen. Dadurch kam es in den Builds 49 und 50 zu Problemen.

Von den 28 fehlgeschlagenen Builds waren letztendlich keine auf Fehler der Entwickler zurückzuführen.

In dem Graphen mit den Compiler-Warnungen wird deutlich, welche Builds von Hudson aufgrund von Compiler-Fehlern (rot dargestellt) abgebrochen wurden:

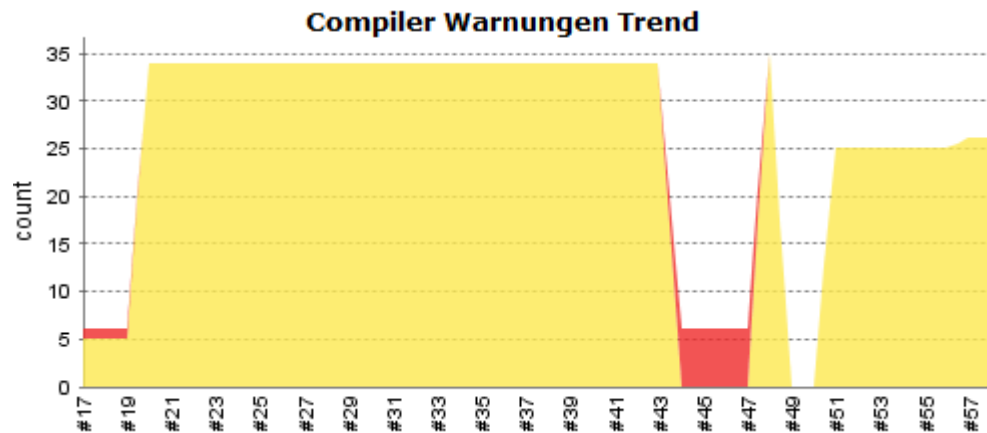


Abbildung 52: C# Projekt - Compiler-Warnungen

In dem Projekt wurde der Quellcodes zusätzlich nach offenen Punkten durchsucht. Der folgende Graph zeigt die Summe der Punkte geringer (blau), mittlerer (gelb) und hoher (rot) Priorität. Die Menge der offenen Punkte hatte sich im Laufe des Projekts nicht verändert.

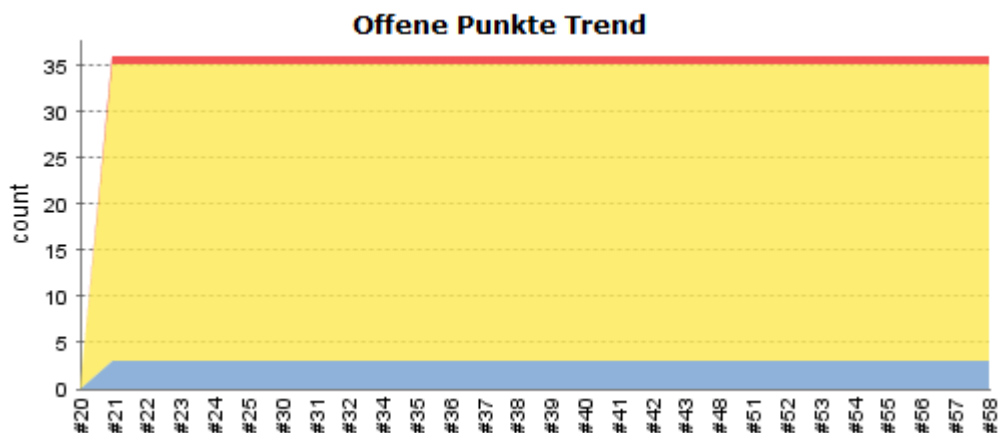


Abbildung 53: C# Projekt - Offene Punkte

5.1.2 VB.NET-Projekt

In diesem Projekt wurde von 4 Software-Entwicklern über einen Zeitraum von 6 Monaten eine Anwendung entwickelt. Der Ist-Aufwand des Projekts war mit 178 PT weit höher als der geplante Aufwand von 65 PT.

Das Projekt beinhaltet zwei Desktop- und eine Webanwendung. Vor dem Einsatz von Hudson war die Release-Erstellung ein manueller Prozess. Der Zeitaufwand für die Erstellung der einzelnen Anwendungen unterscheidet sich dabei leicht. Für die Desktopanwendungen werden Setup-Dateien erstellt und die Webanwendung wird auf

5.1.2 VB.NET-Projekt

einem Test-Server veröffentlicht.

In der folgenden Tabelle ist der Zeitaufwand angegeben, der für die Erstellung der Desktop-Anwendungen entstanden ist:

<u>Vorgang</u>	<u>Zeitaufwand (in Minuten)</u>
Subversion Checkout	1
Visual Studio starten und das Projekt öffnen	5
Kompilierung des Quellcodes	5
Setup-Datei erstellen	5
Zip-Archiv erstellen	2
Summe:	18

Tabelle 5.2 VB.NET-Projekt –Release Zeitaufwand der Desktopanwendung

Die nachfolgende Tabelle zeigt den benötigten Zeitaufwand, der für die Erstellung der Web-Anwendung angefallen ist:

<u>Vorgang</u>	<u>Zeitaufwand (in Minuten)</u>
Subversion Checkout	1
Visual Studio starten und das Projekt öffnen	5
Kompilierung des Quellcodes	5
Kompilat per FTP auf den Server kopieren	5
Summe:	16

Tabelle 5.3 VB.NET-Projekt – Release Zeitaufwand der Webanwendung

Da in diesem Projekt mehr Entwickler tätig waren und viel mehr Änderungen am Quellcode durchgeführt wurden, kam es häufiger zu Problemen als in dem zuvor beschriebenen C#-Projekt. Deshalb wurden beim VB.NET-Projekt bei der Release-Erstellung pauschal 30 Minuten für Fehlersuche und -behebung hinzugefügt. Ein weiterer Grund für diese höhere Pauschale ist, dass bei diesem Projekt auch extern arbeitende Mitarbeiter tätig waren, durch deren Einsatz fehlerhafte checkins immensen Zeitaufwand mit sich führten.

Innerhalb des Projektzeitraumes wurden von den Desktopanwendungen 30 und von der Webanwendung 20 Releases durchgeführt.

Dadurch ergeben sich die folgenden Aufwände für die Desktop- und die Webanwendung:

$$30 * (18 + 30) \text{ Minuten} = 1.440 \text{ Minuten} = 3 \text{ PT}$$

Text 7: VB.NET-Projekt - Release-Aufwand für die Desktopanwendung

$$20 * (16 + 30) \text{ Minuten} = 920 \text{ Minuten} \sim 2 \text{ PT}$$

Text 8: VB.NET-Projekt - Release-Aufwand für die Webanwendung

In dem Projekt wurden demnach fast 5 PT alleine für die Erstellung der Releases eingespart, was ca. 3% des Gesamtaufwandes ausmacht.

Seit der Einrichtung des Projekts wurden in Hudson insgesamt 573 Builds ausgelöst und 202 davon sind fehlgeschlagen. Aufgrund von Umstellungen des Server-Bereichs im Netzwerk der Firma kam es für 14 Tage zu fehlerhaften Builds da Hudson den Server mit der Versionsverwaltung nicht erreichen konnte. Davon waren 165 Builds der Nummern 199 bis 364 betroffen. Daraus resultiert, dass 37 fehlgeschlagene Builds auf Fehler der Entwickler zurückzuführen sind. Dazu gehörten Fehler im Quellcode, vergessene Dateien oder nicht hinzugefügte Bibliotheken.

5.1.2 VB.NET-Projekt

Der folgende Graph zeigt die Dauer der einzelnen Builds, dabei werden die fehlgeschlagenen in rot und die erfolgreichen in blau dargestellt:

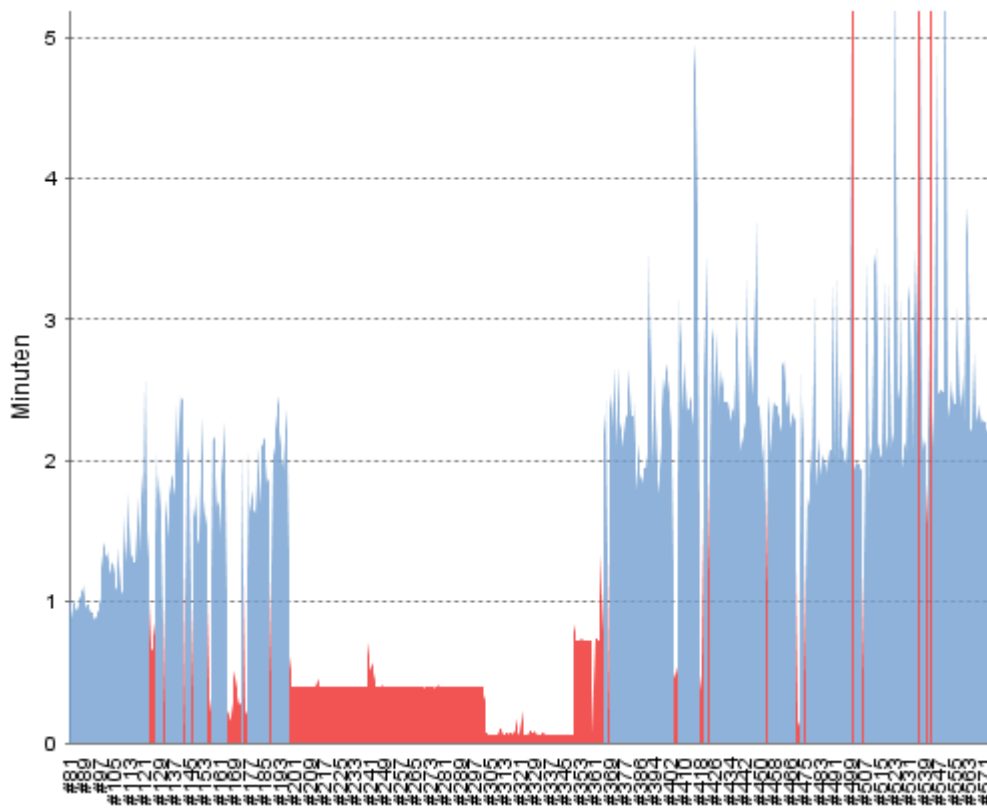


Abbildung 54: VB.NET-Projekt - Build-Dauer

Der folgende Graph stellt die Anzahl der ausgeführten UnitTests dar, der Anteil an fehlgeschlagenen Tests wird in rot dargestellt:

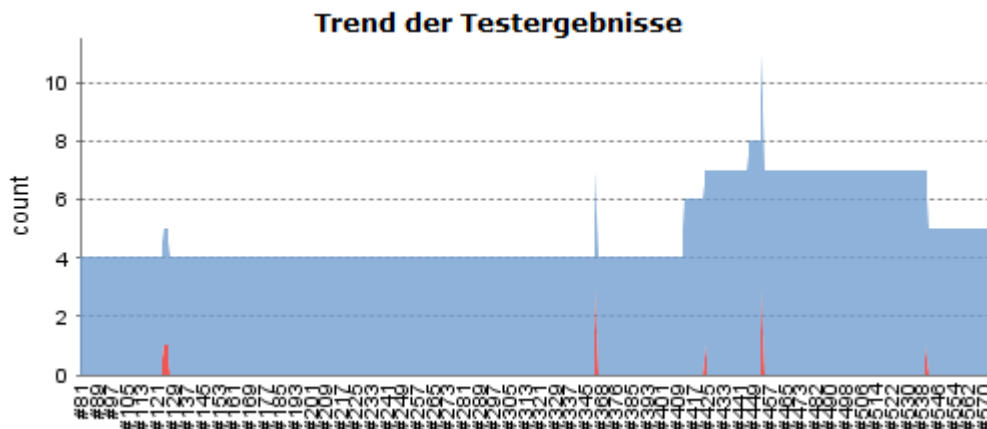


Abbildung 55: VB.NET-Projekt - UnitTests

In der Abbildung 55 wird angezeigt, dass in den Builds 199 bis 364 angeblich Tests ausgeführt wurden. Dies ist jedoch falsch, denn die Tests dieser Builds zeigen das Ergebnis von Build Nummer 198 an. Dies ist auf ein Problem bei der Ausführung der

Jobs zurückzuführen. Die Prüfung der Versionsverwaltung gehört zu den ersten Pre-Build-Aktionen, die in einem Job ausgeführt werden. Die UnitTests werden beim Build-Prozess gestartet und der Bericht mit den Testergebnissen wird vom Plugin „NUnit“ als einer der Post-Build-Aktionen am Ende des Jobs ausgelesen. Kommt es vor der Ausführung der Tests zu einem Fehler wird der Testbericht nicht aktualisiert und das Plugin liest den Bericht des letzten Builds aus. Bei der Umsetzung des Projekts wurde dieses Problem teilweise dadurch behoben, dass man in dem ersten Build-Schritt das Arbeitsverzeichnis des Jobs aufräumt und dabei alte Berichte löscht. Wird ein Build durch einen Fehler in den Pre-Build-Aktionen (wie etwa der Abfrage der Versionsverwaltung) abgebrochen, bleibt das Problem jedoch bestehen.

5.1.2 VB.NET-Projekt

Bei den Compiler-Warnungen war insgesamt ein sehr starker Rückgang zu verzeichnen, was in der folgenden Abbildung erkenntlich wird:

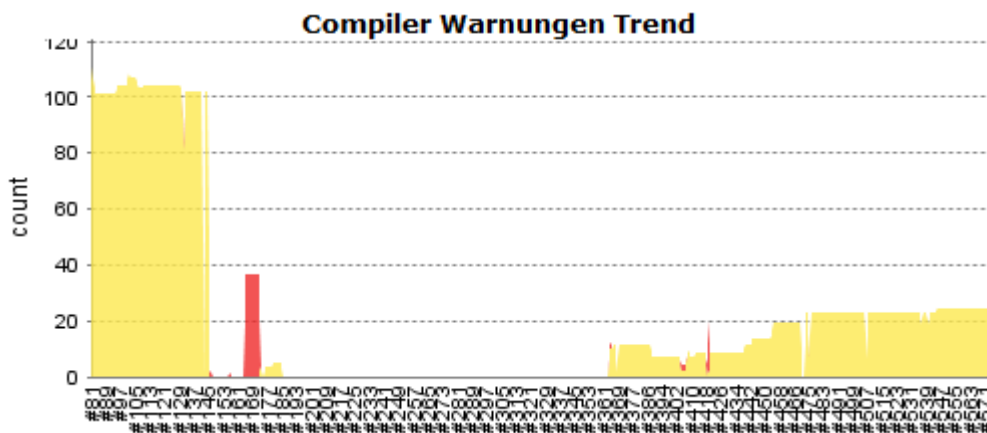


Abbildung 56: VB.NET-Projekt - Compiler Warnungen

Die bereits erwähnten fehlerhaften Builds 199-364 zeigen wie zu erwarten keine Warnungen an, da der Build vor dem Compiler-Vorgang abgebrochen wurde. Der Job wurde so konfiguriert, dass das Plugin „Warnings“ auch bei fehlerhaften Builds im Anschluss die Compiler-Warnungen ausliest. Seit dem Build 146 hatten die Entwickler insbesondere auf die Beseitigung aller Warnungen geachtet. Davor bezogen sich fast alle Warnungen auf nicht verwendete oder nicht initialisierte Variablen. Eine der Warnungen bezog sich auf eine Funktion, die nicht für alle Codepfade einen Wert lieferte. Diese Arten von Warnungen traten auch im späteren Projektverlauf immer wieder auf und wurden meistens in folgenden Builds beseitigt. Die Warnungen hoher Priorität in den Builds 166-173 kamen durch fehlende Assembly-Verweise zustande. Andere Warnungen hoher Priorität traten aufgrund nicht deklarerter Variablen auf, die teilweise auf fehlende Quellcode-Dateien oder Bibliotheken zurückzuführen waren.

Der Graph mit den offenen Punkten zeigt ein gemischtes Bild. Die Punkte wurden nur bei erfolgreichen Builds zusammengefasst, so tauchen z.B. die erwähnten fehlgeschlagenen Builds 199-364 in dem Graphen nicht auf.

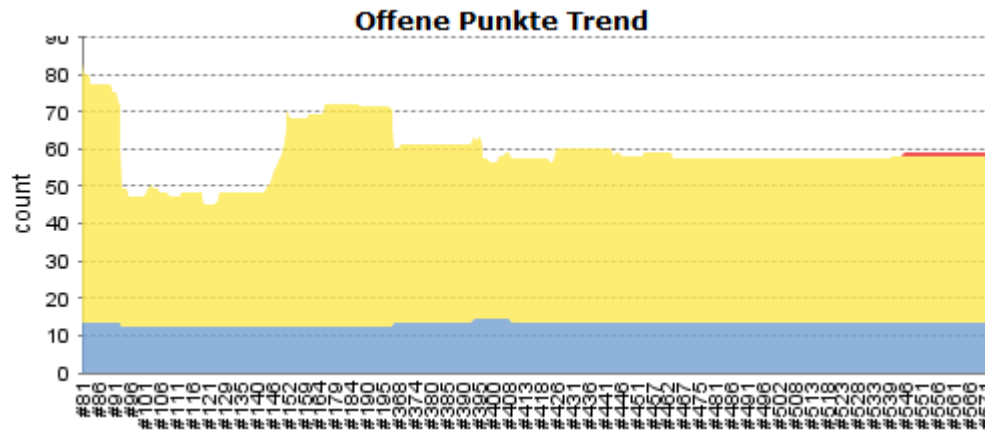


Abbildung 57: VB.NET-Projekt - Offene Punkte

Mit Build 94 wurden viele der Punkte überprüft und beseitigt. Durch die Entwicklung neuer Funktionen stiegen die offenen Punkte ab #145 wieder an und wurden später wieder stufenweise abgebaut. Mit #546 kam ein Punkt mit hoher Priorität hinzu, um eine problematische Stelle im Quellcode zu kennzeichnen. Bei den offenen Punkte handelt es sich um Kommentare, die von den Entwicklern in den Quellcode eingefügt wurden. Das Plugin „Task Scanner“ sucht nach definierten Schlüsselwörtern und fügt den Text zwischen Schlüsselwort und Zeilenende den offenen Punkten hinzu. Da das Plugin aber nicht zwischen Code- und Kommentarzeilen unterscheiden kann, kommt es unter Umständen zu falschen Einträgen. In diesem Projekt gehörten zu den Schlüsselwörtern unter anderem die Texte „REVIEW“ und „TO DO“. Gleichzeitig wurde das Plugin so eingestellt, dass es die Groß-/Kleinschreibung ignoriert. Diese Einstellung wurde vorgenommen, um sicher zu gehen, dass auch andere Schreibweisen der Schlüsselwörter erkannt werden. Es hat sich herausgestellt, dass diese Einstellung eher von Nachteil sein kann. Das Plugin fand Variablen mit dem Namen „review“ und Kommentare in denen der Text „to do“ vorkam und legte damit fälschlicherweise offene Punkte an. So wurden im aktuellsten Build Nummer 573 19 von 59 offenen Punkten falsch angelegt. Darunter befanden sich 13 Punkte niedriger Priorität und 6 Punkte mittlerer Priorität. Dieses Problem lässt sich am besten dadurch lösen, dass man für die Schlüsselwörter eine Schreibweise festlegt an die sich die Entwickler halten müssen und das Plugin so einstellt, dass es die Groß-/Kleinschreibung beachtet.

5.1.3 PHP-Projekt

In diesem Projekt wird für einen Kunden mit Hilfe der Standardsoftware Oxid eShop ein OnlineShop in der Programmiersprache PHP entwickelt. Die Implementierung der Anwendung wurde von 3 Mitarbeitern etwa einen Monat vor dem Abgabetermin dieser Diplomarbeit begonnen.

In der kurzen Zeit kam es deshalb noch zu keinen Releases der Anwendung. Allerdings wurde für das Projekt ein Server eingerichtet, auf dem zu Test- und Demonstrationszwecken die Anwendung installiert wird. Dieser Vorgang wurde durch Hudson ebenfalls automatisiert und erspart den Entwicklern dadurch etwas Arbeitszeit.

Für eine Installation der Anwendung auf dem Test-Server wären normalerweise folgende manuellen Schritte notwendig:

Vorgang	Zeitaufwand (in Minuten)
Subversion Checkout	1
Ausführen von UnitTests	1
Kopieren der Dateien auf den Server	5
Summe:	7

Tabelle 5.4 PHP Projekt – Zeitaufwand zur Installation der Anwendung

Durch die Automatisierung der Installation sparen die Entwickler also bereits 7 Minuten.

In dem betrachteten Projektzeitraum wurde von den Entwicklern bisher noch kein eigener Quellcode erstellt. Alle Änderungen im Quellcode beschränkten sich auf das Anlegen und Anpassen von Konfigurationsdateien für die verwendete Online-Shop Software Oxid eShop.

Insgesamt wurden für dieses Projekt 17 Builds ausgelöst, um die Konfiguration des Hudson Jobs und die Übertragung der Dateien auf den Test-Server zu testen. Dabei schlugen 5 Builds fehl.

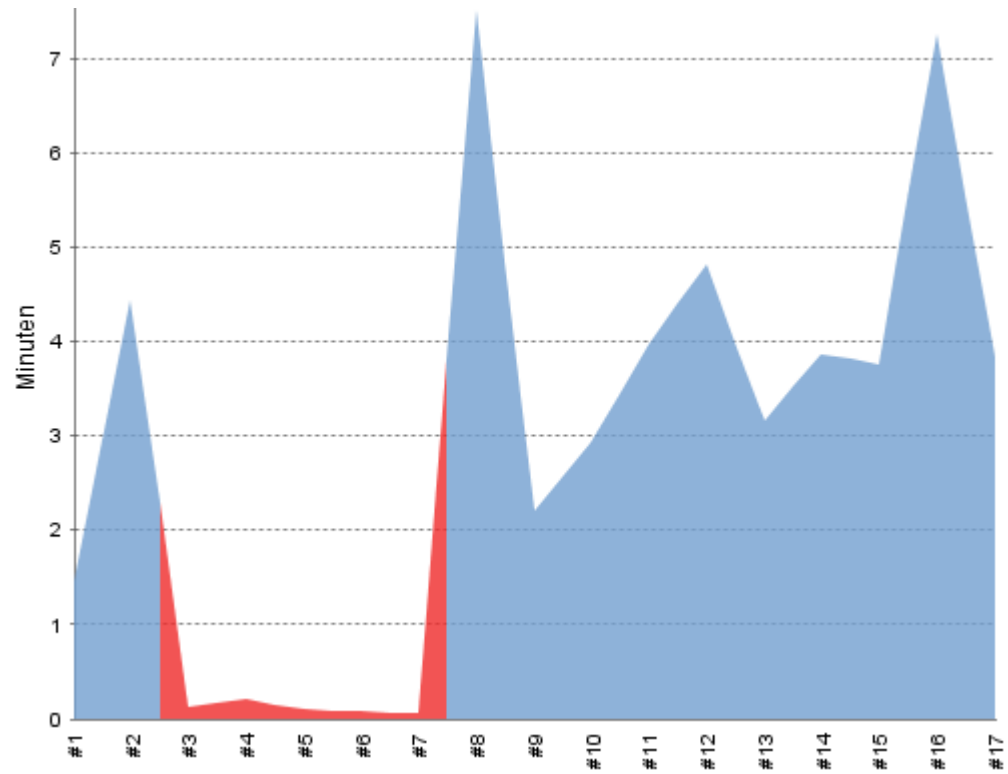


Abbildung 58: PHP Projekt - Build-Dauer

6 Zusammenfassung

Der Einsatz eines Continuous Integration Servers (CIS) ist auch für kleinere Unternehmen interessant. Aktuelle Systeme sind inzwischen sehr benutzerfreundlich und lassen sich einfach konfigurieren und verwalten.

Durch die gezielte Evaluation aktueller CIS nach den Bedürfnissen des Unternehmens ist es möglich das optimale System für das Unternehmen zu ermitteln.

Selbst die kostenlosen CIS erfüllen die meisten Anforderungen, die an einen modernen CIS gestellt werden können. Im Laufe der Arbeit wurde der kostenlose CIS Hudson zur Verwaltung mehrerer Projekte der Firma eWorks GmbH eingesetzt.

In den Projekten konnte alleine durch die Automatisierung der Erstellung der Releases 3% bis 3,75% Arbeitszeit (Im Vergleich zum tatsächlichen Arbeitsaufwand in den Projekten) eingespart werden. Dies ist aber nur der Zeitaufwand, der gemessen werden konnte. Durch den CIS wurde auch die Bereitschaft zu Checkins erhöht und die Entwickler achteten darauf, dass der aktuelle Stand in der Versionsverwaltung auch lauffähig ist.

Für zukünftige Arbeiten wäre die Implementierung eines Ein-Klick-Verfahrens zur Einrichtung neuer Projekte interessant, bei dem die Einrichtung von Repositories in der Versionsverwaltung und das Anlegen der Projekte im CIS automatisiert abläuft.

Weiterhin wird in eWorks auch die Veröffentlichung der Testergebnisse und Build-Artefakte aus dem CIS in Betracht gezogen. Dabei ist auch die automatische Erstellung von Testumgebungen interessant, in dem die Anwendung von Interessenten getestet werden kann.

7 Abkürzungen

CI = Continuous Integration

CIS = Continuous Integration Server

CLI = Command Line Interface

CSS = Cascading Stylesheet

DMZ = Demilitarized Zone (deutsch: entmilitarisierte Zone)

DSL = Digital Subscriber Line

FTP = File Transfer Protocol

HTML = Hypertext Markup Language

IP = Internet Protocol

JDK = Java Development Kit

JRE = Java Runtime Environment

JVM = Java Virtual Maschine

KB = Kilobyte

GB = Gigabyte

GUI = Graphical User Interface

GUID = ID welche zur Referenzierung von Visual Studio Projekten in den Solution-Dateien verwendet wird

LDAP = Lightweight Directory Access Protocol

MD5 = Message-Digest algorithm 5

MS = Microsoft

MB = Megabyte

PC = Personal Computer

PT = Personen Tage

RSS = Really Simple Syndication (seit RSS Version 2.0)

SFTP = Secure File Transfer Protocol

SSH = Secure Shell

SVN = Subversion (Kurzform für die Versionsverwaltung)

URL = Uniform Resource Locator

VB = Visual Basic

VPN = Virtual Private Network

XML = Extensible Markup Language

7 Abkürzungen

z.B. = zum Beispiel

ZIP = Das ZIP-Dateiformat ist ein offenes Format zur komprimierten Archivierung von Dateien. Der Name ZIP sollte damals bei der Veröffentlichung andeuten, dass es schneller als andere Kompressionsformate der Zeit arbeitete.

8 Glossar

Agent = Programm, das von einem CIS zur Ausführung verteilter Builds verwendet wird

Build = **Erstellungsprozess** oder **Build-Prozess** (von englisch *to build* „etwas bauen“) bezeichnet in der Programmierung einen Vorgang, durch den ein fertiges Anwendungsprogramm automatisch erzeugt wird.

CI-Prozess = Beschreibt den von einem CIS durchgeführten Prozess des neu bildens und testens einer Anwendung

Quellcode = Programmcode einer Anwendung

Job = Im CIS Hudson werden die CI-Prozesse mit Hilfe von Jobs konfiguriert. Die Jobs enthalten alle Informationen, die zur Durchführung der Prozesse notwendig sind und legen die einzelnen Build-Schritte fest.

Projekt = Der Begriff Projekt bezieht sich auf ein im Unternehmen durchgeführtes Projekt.

Lable = englisch für Kategorisierung

OpenSource = frei verfügbare Software, dessen Quellcode ebenfalls erhältlich ist.

Post-Build = Zeitlich nach einen Build

Pre-Build = Zeitlich vor einem Build

Release = Veröffentlichung einer Anwendung zur Bereitstellung für den Kunden

Repository = Verwaltetes Archiv in einer Versionsverwaltung

RSS-Feed = Bezeichnet ein XML-Dokument, welches zur Veröffentlichung von Texten eingesetzt wird und neben dem Text (oder einer Zusammenfassung) auch Metadaten (z.B. Datum der Veröffentlichung und Author) enthält.

Snapshots = Abbild einer Systemumgebung

Standalone = (englisch für unabhängig) Eine Standalone-Software benötigt keine weitere Software um lauffähig zu sein.

Versionsverwaltung = Software zur Verwaltung und Versionierung von Dateien

Wiki = Foren-System zur Sammlung von Informationen

Setup-Datei = Archiv mit Installationsprogramm zur Installation der darin enthaltenen Anwendung.

9 Literaturverweise

Martin Fowler, Continuous Integration, 01.05.2006, Internetseite
<http://www.martinfowler.com/articles/continuousIntegration.html>

Paul M. Duvall, Continuous Integration, 2007
ISBN-10: 0-321-33638-0
<http://www.integratebutton.com>

Mike Clark, Projekt-Automatisierung, 2006
ISBN: 3-446-40008-7

Fachzeitschrift dot.NET, Ausgabe Nr. 2.2010 (www.dotnet-magazin.de)
Artikel „Effiziente Frühwarnsysteme“ über Unit-Tests (Seite 44)

Kent Beck, Test Driven Development. By Example, 2002
ISBN 0321146530

Gerard Meszaros, xUnit Test Patterns: Refactoring Test Code, 2007
ISBN 0131495054

10 Quellen

Alternativen zu CruiseControl

<http://confluence.public.thoughtworks.org/display/CC/Understanding+the+alternatives+to+CruiseControl>

Begriff Continuous Integration in Wikipedia

http://de.wikipedia.org/wiki/Continuous_Integration

CI Vergleichstabelle

<http://confluence.public.thoughtworks.org/display/CC/CI+Feature+Matrix>

Continuous integration with Hudson – JavaWorld

<http://www.javaworld.com/javaworld/jw-12-2008/jw-12-hudson-ci.html?page=1>

Hudson Wiki Seite

<http://wiki.hudson-ci.org/display/HUDSON/Home>

Nabble - Hudson forum

<http://www.nabble.com/Hudson-f16871.html>

Publish Web Application via MSBUILD - ASP.NET Forums

<http://forums.asp.net/t/1063636.aspx>

phpunit with Hudson CI

<http://stackoverflow.com/questions/518083/how-might-i-integrate-phpunit-with-hudson-ci>

Setting up continuous integration for PHP using Hudson and Phing « Dave Gardner – PHP Developer

<http://www.davegardner.me.uk/blog/2009/11/09/continuous-integration-for-php-using-hudson-and-phing/>

PHP and Hudson

<http://toptopic.wordpress.com/2009/02/26/php-and-hudson/>

Better SCM Initiative (Sammlung von Informationen zu Versionsverwaltungen)

<http://better-scm.berlios.de/>

11 Abbildungsverzeichnis

Abbildungsverzeichnis

Abbildung 1: Bamboo - Gesamtübersicht.....	37
Abbildung 2: Hudson - Gesamtübersicht.....	37
Abbildung 3: Bamboo - Projekt Zusammenfassung.....	37
Abbildung 4: Hudson - Projekt-Zusammenfassung.....	38
Abbildung 5: Bamboo - Report Parameter.....	38
Abbildung 6: Hudson - Konfiguration der Build-Schritte.....	40
Abbildung 7: Hudson - Hilfe in der Konfiguration.....	40
Abbildung 8: Plugin Seleniumhq - Projekt.....	47
Abbildung 9: Plugin Seleniumhq - Build.....	48
Abbildung 10: Plugin Release - Build Verlauf.....	49
Abbildung 11: Plugin Release - Dashboard Übersicht.....	49
Abbildung 12: Plugin Email-ext - Globale Konfiguration.....	49
Abbildung 13: Plugin Email-ext - Job Konfiguration.....	50
Abbildung 14: Plugin Violations - Graph.....	52
Abbildung 15: Plugin Warnings - Job Konfiguration.....	53
Abbildung 16: Plugin Warnings - Erweiterte Job Konfiguration.....	53
Abbildung 17: Plugin Warnings - Erweiterte Job Konfiguration (Fortsetzung).....	54
Abbildung 18: Plugin Warnings - Build Zusammenfassung.....	54
Abbildung 19: Plugin Warnings - Projekt-Graph.....	55
Abbildung 20: Plugin Task Scanner - Job Konfiguration.....	55
Abbildung 21: Plugin Disk Usage - Projekt Übersicht.....	56
Abbildung 22: Plugin Disk Usage - Build-Verlauf.....	56
Abbildung 23: Plugin Disk Usage - Gesamt-Übersicht.....	56
Abbildung 24: Firefox Addon Hudson Build Monitor.....	58
Abbildung 25: Plugin CI-Game - Build.....	61
Abbildung 26: Plugin CI-Game - Build Details - Punkte.....	61
Abbildung 27: Plugin CI-Game - Build Details - Benutzer.....	61
Abbildung 28: Plugin CI-Game - Rangliste.....	61
Abbildung 29: Link zur Plugin-Verwaltung.....	61
Abbildung 30: Aktualisierung von Plugins.....	62
Abbildung 31: Verfügbare Plugins.....	62
Abbildung 32: Installierte Plugins.....	62
Abbildung 33: Erweiterte Einstellungen der Plugin-Verwaltung.....	63
Abbildung 34: Hudson - Konfiguration - Variablen.....	72
Abbildung 35: C#-Projekt - Konfiguration - Benutzerrechte.....	73
Abbildung 36: C#-Projekt - Konfiguration - Parameter.....	74
Abbildung 37: C#-Projekt - Konfiguration - Timeout.....	75
Abbildung 38: C#-Projekt - Konfiguration - MSBuild.....	75
Abbildung 39: C#-Projekt - Konfiguration - Artefakt.....	77
Abbildung 40: C#-Projekt - Konfiguration - Offene Punkte.....	77
Abbildung 41: VB.NET Projekt – Haupt-Job Konfiguration - Benutzerrechte.....	79
Abbildung 42: VB.NET-Projekt – Haupt-Job Konfiguration - Parameter.....	79
Abbildung 43: VB.NET-Projekt – Haupt-Job Konfiguration - MSBuild.....	80

Abbildung 44: VB.NET-Projekt – Haupt-Job Konfiguration - Artefakte.....	81
Abbildung 45: VB.NET-Projekt – Haupt-Job Konfiguration - offene Punkte.....	81
Abbildung 46: VB.NET-Projekt - Web-Job Konfiguration - MSBuild.....	82
Abbildung 47: VB.NET-Projekt - Test-Server-Job Konfiguration - Knoten.....	83
Abbildung 48: VB.NET-Projekt - Test-Server-Job Konfiguration - Benutzerrechte.....	83
Abbildung 49: VB.NET-Projekt - Test-Server-Job Konfiguration - Parameter.....	84
Abbildung 50: PHP-Projekt - Konfiguration - Benutzerrechte.....	85
Abbildung 51: C# Projekt - Build-Dauer.....	92
Abbildung 52: C# Projekt - Compiler-Warnungen.....	93
Abbildung 53: C# Projekt - Offene Punkte.....	93
Abbildung 54: VB.NET-Projekt - Build-Dauer.....	96
Abbildung 55: VB.NET-Projekt - UnitTests.....	96
Abbildung 56: VB.NET-Projekt - Compiler Warnungen.....	98
Abbildung 57: VB.NET-Projekt - Offene Punkte.....	99
Abbildung 58: PHP Projekt - Build-Dauer.....	101

12 Weitere Verzeichnisse

Tabellenverzeichnis

Tabelle 2.1: Evaluation - Gewichtung.....	27
Tabelle 2.2: Evaluation - Gesamtübersicht.....	31
Tabelle 2.3: Evaluation - Rangliste mit Gesamtpunktzahl (Durchschnitt 6,26 Punkte).....	32
Tabelle 2.4: Evaluation - Detaillierte Punktetabelle der Top 3.....	34
Tabelle 2.5: Evaluation Lizenzkosten.....	36
Tabelle 5.1 C# Projekt – Release-Zeitaufwand.....	91
Tabelle 5.2 VB.NET-Projekt –Release Zeitaufwand der Desktopanwendung.....	94
Tabelle 5.3 VB.NET-Projekt – Release Zeitaufwand der Webanwendung.....	94
Tabelle 5.4 PHP Projekt – Zeitaufwand zur Installation der Anwendung.....	100
Tabelle 13.1: Detaillierte Ergebnisse aus der Evaluation der CIS.....	121

Textverzeichnis

Text 1: Evaluation - Bewertungs-Formel mit Beispiel.....	25
Text 2: Plugin CI-Game - Hinzufügen von Regeln.....	60
Text 3: Plugin Verzeichnisstruktur.....	64
Text 4: Tomcat Port-Einstellung.....	67
Text 5: Tomcat JVM-Einstellung.....	68
Text 6: C# Projekt - Release-Zeitaufwand.....	91
Text 7: VB.NET-Projekt - Release-Aufwand für die Desktopanwendung.....	95
Text 8: VB.NET-Projekt - Release-Aufwand für die Webanwendung.....	95

13 Anhang

13.1 Ergebnisse der Evaluation

	CruiseControl	BuildBot	Anthill	Continuum	Hudson	CruiseControl .Net	CI Factory	Cruise	Anthill Pro	Bamboo	Pulse	TeamCity	BuildBeat CI	OpenMake Meister	OpenMake Mojo	FinalBuilder Server
Code																
Quellcode erhältlich	0,16	0,16	0,16	0,16	0,16	0,16	0,16	0	0	0,16	0	0	0	0	0	0
Redistributio n erlaubt	0,16	0,16	0	0,16	0,16	0,16	0,16	0	0	0	0	0	0	0	0	0
Plugin Support	0,48	0	0	0,48	0,48	0,48	0,48	0,48	0,48	0,48	0,48	0,48	0	0,48	0,48	0,48
Versionsver waltung:																
CVS	0,05	0,05	0,05	0,05	0,05	0,05	0	0	0,05	0,05	0,05	0,05	0,05	0,05	0,05	0,05
Subversion	0,65	0,65	0,65	0,65	0,65	0,65	0,65	0,65	0,65	0,65	0,65	0,65	0,65	0,65	0,65	0,65
Multiple Repositories pro Projekt	0,15	0,15	0	0	0,15	0,15	0	0,15	0,15	0,15	0	0,15	0,15	0,15	0,15	0,15
Dateifilter	0,15	0,15	0	0	0,15	0,15	0,15	0,15	0,15	0,15	0,15	0,15	0,15	0,15	0,15	0,15
Build:																
Parallele Builds	0,48	0,48	0	0,48	0,48	0,48	0,48	0,48	0,48	0,48	0,48	0,48	0,48	0,48	0,48	0,48
Verteilte Builds	0,16	0,16	0	0,16	0,16	0	0	0,16	0,16	0,16	0,16	0,16	0,16	0,16	0,16	0
Auto-Update Agent Code	0	0,08	0	0	0,08	0	0	0,08	0,08	0,08	0,08	0,08	0	0,08	0,08	0

13.1 Ergebnisse der Evaluation

	CruiseControl	BuildBot	Anthill	Continuum	Hudson	CruiseControl .Net	CI Factory	Cruise	Anthill Pro	Bamboo	Pulse	TeamCity	BuildBeat CI	OpenMake Meister	OpenMake Mojo	FinalBuilder Server
Abhängige Builds	0	0,08	0,08	0	0,08	0	0	0,08	0,08	0,08	0,08	0,08	0	0,08	0,08	0,08
Build-Auslöser:																
Versionskontrollsystem	0,4	0,4	0	0	0,4	0	0,4	0,4	0,4	0,4	0,4	0	0,4	0,4	0,4	0
Zeitgesteuert	0,3	0,3	0,3	0,3	0,3	0,3	0,3	0,3	0,3	0,3	0,3	0,3	0,3	0,3	0,3	0,3
manuell	0,3	0,3	0,3	0,3	0,3	0,3	0,3	0,3	0,3	0,3	0,3	0,3	0,3	0,3	0,3	0,3
Sicherheit:																
Benutzerverwaltung	0,3	0	0	0,3	0,3	0	0,3	0,3	0,3	0,3	0,3	0,3	0,3	0,3	0,3	0,3
LDAP	0	0	0	0,2	0,2	0	0,2	0,2	0,2	0,2	0,2	0,2	0,2	0,2	0,2	0,2
Veröffentlichung:																
Email	0,5	0,5	0,5	0,5	0,5	0,5	0,5	0,5	0,5	0,5	0,5	0,5	0,5	0,5	0,5	0,5
FTP	0,2	0	0	0	0,2	0	0	0	0	0,2	0	0	0	0,2	0,2	0,2
SCP	0,2	0	0	0	0,2	0	0	0	0,2	0,2	0	0	0	0,2	0,2	0
System Tray	0,1	0	0	0	0,1	0,1	0,1	0,1	0,1	0,1	0,1	0,1	0,1	0	0	0,1
CLI:																
Build kill	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Build pause	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	CruiseControl	BuildBot	Anthill	Continuum	Hudson	CruiseControl .Net	CI Factory	Cruise	Anthill Pro	Bamboo	Pulse	TeamCity	BuildBeat CI	OpenMake Meister	OpenMake Mojo	FinalBuilder Server
GUI:																
Projekt hinzufügen	0,18	0	0	0	0	0	0	0	0	0	0	0	0,18	0,18	0,18	0
Projekt ändern	0,12	0	0	0	0	0	0	0	0	0	0	0	0,12	0,12	0,12	0
Projekt kopieren	0,06	0	0	0	0	0	0	0	0	0	0	0	0,06	0,06	0,06	0
Verwaltung mehrerer Projekte	0	0	0	0	0	0	0	0	0	0	0	0	0,18	0,18	0,18	0
Agenten verwalten	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Build kill	0	0	0	0	0	0	0	0	0	0	0	0	0,36	0,36	0,36	0
Build pause	0	0	0	0	0	0	0	0	0	0	0	0	0	0,06	0,06	0
Zugriff auf Build-Dateien	0	0	0	0	0	0	0	0	0	0	0	0	0	0,06	0,06	0
Build Statistik	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Web-Oberfläche:																
Projekt hinzufügen	0	0	0,18	0,18	0,18	0	0	0,18	0,18	0,18	0,18	0,18	0	0	0	0,18
Projekt ändern	0	0	0,12	0,12	0,12	0	0	0	0,12	0,12	0,12	0,12	0	0	0	0,12
Projekt	0	0	0	0	0,06	0	0	0	0,06	0,06	0,06	0,06	0	0	0	0,06

13.1 Ergebnisse der Evaluation

	CruiseControl	BuildBot	Anthill	Continuum	Hudson	CruiseControl .Net	CI Factory	Cruise	Anthill Pro	Bamboo	Pulse	TeamCity	BuildBeat CI	OpenMake Meister	OpenMake Mojo	FinalBuilder Server
kopieren																
Verwaltung mehrerer Projekte	0,18	0	0,18	0,18	0,18	0,18	0,18	0,18	0,18	0,18	0,18	0,18	0	0	0	0,18
Agenten verwalten	0	0	0	0,06	0,06	0	0	0,06	0,06	0,06	0,06	0,06	0	0	0	0,06
Build kill	0,36	0,36	0	0,36	0,36	0	0	0,36	0,36	0,36	0,36	0,36	0	0	0	0,36
Build pause	0,06	0	0	0	0	0,06	0	0,06	0,06	0	0,06	0,06	0	0	0	0,06
Zugriff auf Build-Dateien	0,06	0,06	0,06	0,06	0,06	0	0,06	0,06	0,06	0,06	0,06	0,06	0	0	0	0
Build Statistik	0,12	0,12	0	0	0,12	0	0	0	0,12	0,12	0,12	0,12	0,12	0,12	0,12	0,12
Darstellung von Test-Ergebnissen :																
JUnit	0,1	0	0	0,1	0,1	0,1	0,1	0,1	0,1	0,1	0,1	0,1	0	0,1	0,1	0
NUnit	0	0	0	0	0,35	0,35	0,35	0,35	0,35	0,35	0,35	0,35	0,35	0,35	0,35	0,35
PHPUnit	0	0	0	0	0	0	0	0,2	0	0,2	0	0	0	0	0	0
Selenium	0	0	0	0	0,2	0	0	0,2	0,2	0	0	0,2	0	0	0	0
MS Test	0	0	0	0	0	0	0,15	0	0	0	0	0	0	0	0	0
Direkter Build-Tool Support:																

	CruiseControl	BuildBot	Anthill	Continuum	Hudson	CruiseControl .Net	CI Factory	Cruise	Anthill Pro	Bamboo	Pulse	TeamCity	BuildBeat CI	OpenMake Meister	OpenMake Mojo	FinalBuilder Server
Shell / cmd	0,4	0,4	0	0,4	0,4	0,4	0,4	0,4	0,4	0,4	0,4	0,4	0,4	0,4	0,4	0,4
Ant	0,1	0	0,1	0,1	0,1	0	0,1	0,1	0,1	0,1	0,1	0,1	0	0,1	0,1	0,1
Maven 1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Maven 2	0,05	0	0	0,05	0,05	0	0	0	0,05	0,05	0,05	0,05	0	0,05	0,05	0
MS Build	0	0	0	0	0,2	0,2	0,2	0	0,2	0,2	0,2	0,2	0,2	0,2	0,2	0,2
Nant	0,1	0	0	0	0,1	0,1	0,1	0,1	0,1	0,1	0	0,1	0,1	0,1	0,1	0,1
devenv (Visual Studio)	0	0	0	0	0	0,1	0,1	0	0,1	0,1	0	0	0,1	0,1	0	0,1
Phing	0	0	0	0	0,05	0	0	0	0	0	0	0	0	0	0	0
Direkter Test-Framework Support:																
JUnit	0	0	0	0	0	0	0	0	0,05	0	0	0	0	0,05	0	0
NUnit	0	0	0	0	0	0	0	0	0,18	0	0	0	0	0,18	0	0
PHPUnit	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Selenium	0	0	0	0	0,1	0	0	0	0,1	0	0	0	0	0	0	0
MS Test	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Summe:	<u>6,63</u>	<u>4,56</u>	<u>2,68</u>	<u>5,35</u>	<u>7,89</u>	<u>4,97</u>	<u>5,92</u>	<u>6,68</u>	<u>7,71</u>	<u>7,68</u>	<u>6,63</u>	<u>6,68</u>	<u>5,85</u>	<u>7,45</u>	<u>7,12</u>	<u>6,33</u>

Tabelle 13.1: Detaillierte Ergebnisse aus der Evaluation der CIS

13.2 Build-Skripte

Hier werden die bei der Umsetzung der Projekte verwendeten Windows Shell-Skripte aufgeführt.

13.2.1 C#-Projekt

Aufräumen des Arbeitsverzeichnisses:

```
[1] @echo off & setlocal
[2] SET DEL_PATTERN=*Web-Client*.zip
[3]
[4] echo.
[5] echo --- Lösche alte Zips (%DEL_PATTERN%) ---
[6] del %WORKSPACE%\%DEL_PATTERN%
[7] echo.
```

Erzeugen des Release-Paketes:

```
[1] @echo off
[2] :: Batch zum Auslesen der Versionsnummer und Starten der
release.bat
[3] setlocal
[4] :: Variablen
[5] ::SET %RELEASE_VERSION%=
[6] SET WORK_DIR=%WORKSPACE%\trunk\work
[7] SET ASSEMBLY_INFO=%WORKSPACE
%\trunk\source\Properties\AssemblyInfo.cs
[8] :: regulärer Ausdruck für die Zeile mit der Versionsnummer:
[9] SET VPATTERN="^[assembly: AssemblyVersion(\".*\")]"
[10]
[11]
[12] :: Auslesen der Versionsnummer
[13] echo.
[14] echo
-----
[15] echo Start des Release Vorgangs
[16] echo.
[17]
[18] IF "%RELEASE_VERSION%" == "" (
[19] echo Lese Datei %ASSEMBLY_INFO% aus
[20] FOR /f "tokens=2 delims=()" %%i IN ('findstr /R /C:%VPATTERN%
%ASSEMBLY_INFO%') DO SET RELEASE_VERSION=%%~i
[21] )
[22]
[23] IF "%RELEASE_VERSION%" == "" call :error "Versionsnummer
nicht gefunden"
[24] echo.
```

```
[25] echo Verwende Versionsnummer "%RELEASE_VERSION%"
[26] echo.
[27]
[28] :: Wechseln ins work Verzeichnis, damit release.bat richtig
funktioniert
[29] cd %WORK_DIR%
[30] :: Aufruf der release.bat
[31] call release.bat %RELEASE_VERSION% 2>&1
[32]
[33] echo.
[34] echo Release erstellt unter %fullDestDir%
[35] echo
-----
[36] echo.
[37]
[38] echo
-----
[39] echo Zip Erstellung
[40] echo
-----
[41] :: übernehme Variablen aus der release.bat
[42] IF "%filename%" == "" (
[43]   call :error "Dateiname für die zip ist leer"
[44] )
[45] SET ZIP_NAME="%filename%.zip"
[46] IF "%fullDestDir%" == "" (
[47]   call :error "Release-Verzeichnis zum zipen nicht gefunden"
[48] )
[49] SET SOURCE_FILES="%fullDestDir%*"
[50]
[51] :: Stelle aktuellen Pfad auf workspace
[52] cd %WORKSPACE%
[53]
[54] echo Erstelle Zip Datei %ZIP_NAME%
[55] echo.
[56] %ZIP7% a %WORKSPACE%\%ZIP_NAME% -tzip %SOURCE_FILES%
[57] echo
-----
[58] echo.
[59]
```

13.2.1 C#-Projekt

```
[60] echo
-----
[61] echo Lösche Releases Verzeichnis
[62] rd /s /q "%WORK_DIR%\Releases"
[63] echo
-----
[64] echo.
[65] GOTO :eof
[66]
[67] :error
[68] echo.
[69] echo ERROR %1
[70] echo.
[71] exit 1
[72] GOTO :eof
```

13.2.2 VB.Net-Projekt

13.2.2.1 Haupt-Job

Aufräumen des Arbeitsverzeichnisses:

```
[1] @echo off & setlocal
[2] SET DEL_PATTERN=ABC*.zip
[3]
[4] echo.
[5] echo --- Lösche alte Artefakte (%DEL_PATTERN%) ---
[6] del %DEL_PATTERN%
[7] echo.
[8] echo --- Lösche alte Test-Berichte ---
[9] del NUnitResults.xml
[10] echo.
```

Ausführen der UnitTests:

```
[1] @echo off
[2] echo --- UnitTests ---
[3] %NUNIT_2_CONSOLE% %WORKSPACE%
%KUNDENNAME%\PROJEKT\source\ABC\ABC.Unittests\bin\Debug\ABC.Unittests.dll /xml=NUnitResults.xml
```

Erstellen der Setup-Dateien:

```
[1] @echo off
[2] echo.
[3] echo --- Erstelle Setup 1 ---
[4] echo --- Startzeit: %TIME% ---
[5] echo.
[6] %DEVENV_VS2008% %WORKSPACE
%\KUNDENNAME\PROJEKT\source\ABC\ABC.sln /rebuild Debug /project
"ABC-Manager Setup" /projectconfig Debug
[7] echo.
[8] echo --- Setup 1 Endzeit: %TIME% ---
[9] echo.
[10] echo --- Erstelle Setup 2 ---
[11] echo --- Startzeit: %TIME% ---
[12] echo.
[13] %DEVENV_VS2008% %WORKSPACE
%\KUNDENNAME\PROJEKT\source\ABC\ABC.sln /rebuild Debug /project
"ABC-Viewer Setup" /projectconfig Debug
[14] echo.
[15] echo --- Setup 2 Endzeit: %TIME% ---
[16] echo.
```

Erstellung der ZIP-Archive:

```
[1] @echo off
[2] echo.
[3] echo --- Erstellen der Zip-Dateien mit 7-zip ---
[4] echo.
[5] set ZIP_MANAGER=ABC-Manager_%ARCHIV%.zip
[6] set DIR_MANAGER=%WORKSPACE%\KUNDENNAME\PROJEKT\source\ABC\ABC-
Setup\Debug
[7] set ZIP_VIEWER=ABC-Viewer_%ARCHIV%.zip
[8] set DIR_VIEWER=%WORKSPACE%\KUNDENNAME\PROJEKT\source\ABC\"ABC-
Viewer Setup"\Debug
[9]
[10] rem Löschen der alten zips als ersten Build-Schritt, da bei
einem Fehlschlag sonst die alten zips archiviert werden
[11] rem -- Manager zip --
[12] ::del %ZIP_MANAGER%
[13] %ZIP7% a %WORKSPACE%\%ZIP_MANAGER% -tzip %DIR_MANAGER%\*.*
[14]
[15] rem -- Viewer Zip --
[16] ::del %ZIP_VIEWER%
[17] %ZIP7% a %WORKSPACE%\%ZIP_VIEWER% -tzip %DIR_VIEWER%\*.*
```

13.2.2.1 Haupt-Job

```
[18]
[19] IF "%ARCHIV%" == "standard" GOTO :ende
[20] echo.
[21] echo --- Füge weitere Dateien hinzu für komplettes Archiv ---
[22] echo.
[23] %ZIP7% a %WORKSPACE%\%ZIP_MANAGER% -tzip %DIR_MANAGER
%\Office2007PIARedist\*.*
[24] %ZIP7% a %WORKSPACE%\%ZIP_VIEWER% -tzip %DIR_VIEWER
%\Office2007PIARedist\*.*
[25]
[26] :ende
```

13.2.2.2 Web-Job

Aufräumen des Arbeitsverzeichnisses:

```
[1] @echo off
[2] echo.
[3] echo --- Lösche %WORKSPACE%\WebPublish ---
[4] rd /s /q %WORKSPACE%\WebPublish
[5] echo.
[6]
[7] echo.
[8] echo --- Lösche alte Zip-Dateien ---
[9] del %WORKSPACE%\ABC*.zip
[10] echo.
```

Erstellen des ZIP-Archivs und Kopieren auf den Test-Server:

```
[1] @echo off & setlocal
[2] set ZIP_NAME=ABC-Web.zip
[3] set WEB_DIR=%WORKSPACE%\WebPublish
[4] set CONFIG_DIR=%WORKSPACE
%\KUNDENNAME\PROJEKT\source\ABC\ABC.Configuration
[5] set CONFIG_DIR_ABC=%CONFIG_DIR%\ABC
[6] set CONFIG_DIR_ABC_STATS=%CONFIG_DIR%\ABC_Stats
[7]
[8] echo -- Füge configs hinzu --
[9] SET WEB_CONFIG_DIR=%WEB_DIR%\config
[10] md %WEB_CONFIG_DIR%
[11] copy /Y %CONFIG_DIR_ABC%\abc.xml %WEB_CONFIG_DIR%
```

```
[12] copy /Y %CONFIG_DIR_ABC%\abc.xsd %WEB_CONFIG_DIR%
[13] copy /Y %CONFIG_DIR_ABC_STATS%\abc_stats.xml %WEB_CONFIG_DIR%
[14] copy /Y %CONFIG_DIR_ABC_STATS%\abc_stats.xsd %WEB_CONFIG_DIR%
[15]
[16] echo.
[17] echo --- Erstelle Zip-Datei %ZIP_NAME% ---
[18] echo.
[19] %ZIP%7 a %WORKSPACE%\%ZIP_NAME% -tzip %WEB_DIR%\*
[20]
[21] echo.
[22] echo --- Kopiere %ZIP_NAME% auf abc ---
[23] echo.
[24] SET ABC_DIR=\\abc\ftproot
[25] net use %ABC_DIR% /user:USERNAME PASSWORD
[26] xcopy /Y %ZIP_NAME% %ABC_DIR%
[27] echo.
```

13.2.2.3 Test-Server-Job

Installation der Webanwendung:

```
[1] @echo off & setlocal
[2] SET ZIP_NAME=ABC-Web.zip
[3] SET SOURCE_DIR=C:\inetpub\ftproot
[4] SET ZIP_FULLNAME=%SOURCE_DIR%\%ZIP_NAME%
[5] SET DEST_DIR=C:\inetpub\wwwroot\PROJEKT
[6] SET BACKUP_WEB=%SOURCE_DIR%\Web.config.%BUILD_NUMBER%
[7] SET PATH_WEB=%DEST_DIR%\Web.config
[8]
[9] echo.
[10] echo -- Sichere alte Web.Config --
[11] echo -- von %PATH_WEB% --
[12] echo -- nach %BACKUP_WEB% --
[13] echo.
[14] copy /Y %PATH_WEB% %BACKUP_WEB%
[15]
[16] echo.
[17] echo -- Lösche Verzeichnis %DEST_DIR% --
[18] del /Y /s /f %DEST_DIR%\*
[19] echo.
[20]
```

13.2.2.3 Test-Server-Job

```
[21] IF EXIST upload.zip SET ZIP_FULLNAME=upload.zip
[22]
[23] rem sichere datei fuer fingerprinting
[24] copy /Y %ZIP_FULLNAME% %WORKSPACE%\lastupload.zip
[25] echo.
[26] echo -- Entpacke Zip %ZIP_FULLNAME% nach %DEST_DIR% --
[27] echo.
[28] %ZIP7% x %ZIP_FULLNAME% -o%DEST_DIR% -y
[29]
[30] echo.
[31] echo -- Stelle alte Web.Config wiederher --
[32] echo.
[33] copy /Y %BACKUP_WEB% %PATH_WEB%
```

Löschen der Upload-Datei:

```
[1] @echo off & setlocal
[2] echo.
[3] echo -- Räume auf --
[4] del upload.zip
[5] echo.
```

13.2.3 PHP-Projekt

Aufruf des Build-Skriptes:

```
[1] phing -f %WORKSPACE%\Web-Shop\build.xml build deploy-test
-Dws=%WORKSPACE%
```

Phing Build-Skript:

```
[1] <?xml version="1.0" encoding="UTF-8"?>
[2] <project name="Web-Shop" basedir="." default="build">
[3]     <property name="builddir" value="{ws}/build" />
[4]     <property name="testsdire" value="{project.basedir}/test" />
[5]     <property name="sourcedir" value="{project.basedir}/src" />
[6]     <property name="testserverdire" value="\\ws_vm\Web-Shop" />
[7]     <property name="copyfilesdire" value="{ws}\Web-Shop\src" />
[8]
[9]     <target name="clean">
[10]         <echo msg="Clean..." />
```

```
[11]     <delete dir="${builddir}" />
[12] </target>
[13]
[14] <target name="prepare">
[15]     <echo msg="Prepare..." />
[16]     <mkdir dir="${builddir}" />
[17]     <mkdir dir="${builddir}/logs" />
[18]     <mkdir dir="${builddir}/logs/coverage" />
[19]     <mkdir dir="${builddir}/docs" />
[20]     <mkdir dir="${builddir}/app" />
[21] </target>
[22]
[23] <!-- app -->
[24] <target name="app">
[25]     <echo msg="Hier passiert noch nichts..." />
[26] </target>
[27]
[28] <!-- Kopiert die Anwendung auf den Testserver -->
[29] <target name="deploy-test">
[30]     <echo msg="Kopiere auf Testserver..." />
[31]     <!-- zugriff auf den testserver -->
[32]     <exec command="net use ${testserverdir}
/user:USERNAME PASSWORD" />
[33]     <!-- kopieren der dateien -->
[34]     <copy todir="${testserverdir}"
[35]         overwrite="false"
[36]         includeemptydirs="true" >
[37]         <fileset dir="${copyfilesdir}">
[38]             <include name="**/*" />
[39]         </fileset>
[40]     </copy>
[41]     <!-- Lösche alte temporäre Dateien auf dem Test-
Server -->
[42]     <echo msg="Lösche Dateien im tmp Verzeichnis..." />
[43]     <delete>
[44]         <fileset dir="${testserverdir}\tmp">
[45]             <include name="**/*" />
[46]         </fileset>
[47]     </delete>
[48] </target>
```

```
[49]
[50]     <!-- PHP API Documentation -->
[51]     <target name="phpdoc">
[52]         <echo msg="PHP Documentor..." />
[53]         <phpdoc title="API Documentation"
[54]             destdir="{builddir}/docs"
[55]             sourcecode="yes"
[56]             defaultpackagename="MHTest"
[57]             output="HTML:Smarty:PHP">
[58]             <fileset dir="{sourcedir}">
[59]                 <include name="**/*.php" />
[60]             </fileset>
[61]         </phpdoc>
[62]     </target>
[63]
[64]     <!-- PHP copy/paste analysis -->
[65]     <target name="phpcpd">
[66]         <echo msg="PHP Copy/Paste..." />
[67]         <exec command="phpcpd --log-pmd=${
[68] {builddir}/logs/pmd.xml {sourcedir}" escape="false" />
[69]     </target>
[70]
[71]     <!-- PHP dependency checker -->
[72]     <target name="pdepend">
[73]         <echo msg="PHP Depend..." />
[74]         <exec command="pdepend --jdepend-xml=${
[75] {builddir}/logs/jdepend.xml {sourcedir}" escape="false" />
[76]     </target>
[77]
[78]         <!-- PHP CodeSniffer -->
[79]     <target name="phpcs">
[80]         <echo msg="PHP CodeSniffer..." />
[81]         <exec command="phpcs --standard=ZEND
[82] --report=checkstyle {sourcedir} > $
[83] {builddir}/logs/checkstyle.xml" escape="false" />
[84]     </target>
[85]
[86]     <!-- Unit Tests & coverage analysis -->
[87]     <target name="phpunit">
[88]         <echo msg="PHP Unit..." />
```

```
[85]         <exec command="phpunit
--log-junit ${builddir}/logs/phpunit.xml
--log-pmd ${builddir}/logs/phpunit.pmd.xml
--coverage-clover ${builddir}/logs/coverage/clover.xml
--coverage-html ${builddir}/logs/coverage/
${testsdire}"/>
[86]     </target>
[87]
[88]     <!-- Main Build target -->
[89]     <target name="build"
[90]         depends="clean, prepare, phpunit"/>
[91]
[92]     <!-- complete Build including doc and metrics -->
[93]     <target name="complete"
[94]         depends="clean, prepare, phpunit, phpdoc, metrics"/>
[95]
[96]     <!-- metric tools -->
[97]     <target name="metrics"
[98]         depends="phpcpd, pdepend, phpcs"/>
[99]
[100] </project>
```