Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Hochschule für Angewandte Wissenschaften Hamburg

Fakultät Life Sciences

Planung und Einführung von Werkzeugen zur Kontinuierlichen Integration im Software-Lebenszyklus

Bachelorarbeit

im Studiengang Medizintechnik

vorgelegt von

Dennis Kock
1958132

Hamburg

am 28. Oktober 2014

Gutachter: Prof. Dr. med. Dipl.-Phys. Jürgen Stettin HAW Hamburg
Gutachter: Dipl.-Ing. Karsten Stegelmann Söring GmbH

Die Bachelorarbeit wurde betreut und erstellt bei der Söring GmbH

Erklärung

Ich versichere hiermit, dass ich die vorliegende Bachelorarbeit mit dem Titel

> *Planung und Einführung von Werkzeugen zur Kontinuierlichen Integration im Software-Lebenszyklus*

ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Quellen und Hilfsmittel verwendet habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich gemacht.

_____

Dennis Kock

Zusammenfassung

Diese Bachelorarbeit befasst sich mit der Auswahl eines geeigneten Werkzeugs, mit dessen Hilfe ein *Continuous Integration* Prozess in der Softwareentwicklungsabteilung der Söring Medizintechnik GmbH ermöglicht werden soll. Dieses Werkzeug soll wesentliche Schritte der Softwareentwicklung automatisiert ablaufen lassen – hierzu gehören der *Buildvorgang* selbst sowie nachgelagerte Tests. Die Anforderungen an die auszuwählenden Werkzeuge sind in einem betriebsinternen Lastenheft definiert worden, dieses stellt den Ausgangspunkt für alle Entscheidungsfindungen im Verlaufe dieser Arbeit dar.
Es wird beschrieben, warum ein sogenannter *Build-Server* als Werkzeug für die vorhandene Umgebung am besten geeignet ist. Es werden sechs verschiedene *Build-Server* auf die Erfüllung der Anforderungen hin untersucht. Mit Hilfe einer Punktwertemethode wird festgestellt, welche vier dieser sechs *Build-Server* näher untersucht werden sollen. Dies geschieht im Rahmen einer Testinstallation, die letztendlich Aufschluss darüber gibt, welche Lösung am besten in den Entwicklungsprozess passen würde.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Glossar und Abkürzungen

CI	Continuous Integration
IDE	Integrated Development Environment
GUI	Graphic User Interface
SMT	Söring Medizintechnik
TFS	(Microsoft) Team Foundation Server
Software-System	Integrierte Sammlung von Software-Komponenten, die so organisiert sind, dass eine spezifische Funktion oder ein Satz von Funktionen ausgeführt werden kann [111, Abschnitt 3 – Begriffe]
(Software-)Komponente	Jedes identifizierbare Teil eines Computerprogramms [111, Abschnitt 3 – Begriffe]
Software-Einheit	Software-Komponente, die nicht in weitere Komponenten unterteilt ist [111, Abschnitt 3 – Begriffe]

1 Einleitung

Im Rahmen dieser Bachelorarbeit sollen Werkzeuge für den Ausbau des Software-Entwicklungsprozesses der Söring Medizintechnik GmbH (SMT) in Richtung eines *Continuous-Integration-Prozesses* (*CI-Prozess*) nach definierten Kriterien und Anforderungen bewertet, ausgewählt und implementiert werden.
Mit diesem CI-Prozess geht eine Automatisierung der verschiedenen Teilschritte bei der Erstellung neuer sowie der Wartung bestehender Software einher – mit fortschreitender Reife des Prozesses nimmt auch der Grad dieser Automatisierung zu. Der erste zu automatisierende Schritt ist hierbei der Buildvorgang mit statischer Code-Analyse, ein weiterer ist die automatisierte Durchführung von Unit Tests. In der letzten denkbaren Ausbaustufe sollen auch Systemtests automatisiert gestartet und ausgewertet werden.

Continuous Integration ist eine mögliche Antwort auf einen an Komplexität zunehmenden Softwareentwicklungsprozess, der dennoch ein Produkt hervorbringen soll, das strengen Qualitätsanforderungen genügen muss.
Immer umfangreichere Softwaresysteme erfordern immer mehr Entwickler, die an verschiedenen Teilen des Endproduktes mitwirken. Hierfür ist es wichtig festzustellen, wo die Gesamtsoftware in kleinere Komponenten zerlegt werden kann. Sind diese sogenannten Softwarekomponenten identifiziert, kann separat an ihnen gearbeitet werden. Diese lose Kopplung der Softwarekomponenten bringt zudem den Vorteil mit sich, dass bestimmte Qualitätsmerkmale – insbesondere die Änderbarkeit, Übertragbarkeit und Wartbarkeit des Codes – einfacher umzusetzen sind.
Auf der anderen Seite bedeutet ein aus vielen Komponenten bestehendes System einen enormen Integrationsaufwand, wenn all diese Komponenten sich an einem Meilenstein zu einem Gesamtsystem zusammenfügen sollen. Hier setzt die Idee der kontinuierlichen Integration an, indem diese Integration inklusive qualitätssichernder Maßnahmen – wie der Auswertung von Metriken[1] und dem Ausführen von Tests – immer häufiger und darüber hinaus automatisiert stattfindet. Kleinere Schritte und dadurch bedingt kleinere Änderungen an den Softwarekomponenten, so der Gedanke, sollen dafür sorgen, dass die bei der nun häufig stattfindenden Integration auftretenden Fehler frühzeitig und isoliert betrachtet und gelöst werden können. In letzter Konsequenz bedeutet

[1] Metriken ordnen diversen Softwareeigenschaften Zahlen zu, um diese bewerten zu können. Typische Metriken sind beispielsweise *Lines of Code*, Kommentardichte, Komplexität

kontinuierliche Integration, dass nach jeder Änderung am Code vollautomatisch gebaut und getestet wird. Neue Fehler lassen sich somit immer auf diese letzte Änderung zurückführen, wodurch sie sehr viel einfacher zu finden und zu beheben sind. Ein weiterer Vorteil, der den Anforderungen moderner Software-Entwicklung entgegen kommt, besteht darin, fortlaufend einen präsentablen Softwarestand bereitzuhalten. Dieser entspricht dem aktuellen Entwicklungsstand, sodass *Stakeholder* einen aktuellen Überblick behalten können und nicht mehr darauf warten müssen, dass beispielsweise ein Meilenstein erreicht wird.
Im ersten Ansatz wäre auch eine etwas schmalere Lösung denkbar: Gewissermaßen als Vorstufe eines vollständigen CI-Prozesses wird beim sogenannten *Nightly Build*, wie der Name schon sagt, jede Nacht gebaut und getestet. Die Integrationsschrittweite ist dann zwar etwas gröber, dennoch sind die Vorteile der kontinuierlichen Integration – wie die Möglichkeit der Zuordnung von Fehlern zu bestimmten Build-Vorgängen sowie den aktuellen Entwicklungsstand widerspiegelnde Software – weitestgehend gegeben.

Es soll also weniger darum gehen, das oftmals und vollmundig gepriesene Gesamtkonzept *Continuous Integration* um seiner selbst willen einer bestehenden Entwicklungsumgebung „überzustülpen“, als vielmehr darum, passende Methoden, die unter dem Begriff CI zusammengefasst werden, für sich zu nutzen und in diese Entwicklungsumgebung zu integrieren. Der Fokus soll hierbei in erster Linie auf der Automatisierung verschiedener Teilschritte – wie dem Buildvorgang, der Code-Analyse und den Tests – liegen. Man könnte daher eher von *automatischer Integration* sprechen. Sind diese Voraussetzungen indes gegeben, ist der Schritt zur *Kontinuierlichen Integration* einfach; denn je leichter und automatisierter die Integration funktioniert, desto öfter und *kontinuierlicher* kann sie auch durchgeführt werden.

Am Ende soll demnach eine Umgebung entstanden sein, in der wesentliche Schritte der Softwareentwicklung automatisiert worden sind und die somit kontinuierliche Integration ermöglicht. Dabei soll die Auswahl des nötigen Werkzeugs bzw. der nötigen Werkzeuge auf Grundlage detaillierter Anforderungen dokumentiert, mit Alternativen verglichen und diskutiert werden. Schlussendlich soll auch die konkrete Umsetzung dokumentiert und deren Erfüllen der Anforderungen in der Praxis ausgewertet werden.

2. Konkrete Anforderungen

Hier sollen die in der Einleitung erwähnten *definierten Kriterien und Anforderungen*, die an die auszuwählenden Werkzeuge gestellt werden, näher erläutert und konkretisiert werden. Diese gehen aus dem Lastenheft LAH-00021-00.05 hervor, in dem die geforderten Anwendungsfälle sowie funktionale und nicht funktionale Anforderungen formuliert sind – aus ihnen ergibt sich auch die im nächsten Kapitel erläuterte Methodik.

2.1. Anwendungsfälle

Anwendungsfall 1: Durchführung und Erzeugung eines/r Software-Systems/-Komponente

Der erste Anwendungsfall beschreibt die Erzeugung eines Softwaresystems oder einer Softwarekomponente. Dies umfasst Codeübersetzungsläufe, Objektarchivierung für Bibliotheken sowie Objektverknüpfungsläufe.
Hierfür wählt der Ausführende das betreffende Softwaresystem oder die betreffende Systemkomponente sowie deren gewünschte Revision aus. Die Erzeugung soll zeitgesteuert gestartet werden. Das Dateisystem wird nun aus der Versionsverwaltung mit den notwendigen Dateien gefüllt und es werden Übersetzungen, Archivierungen und Bindungen durchgeführt. Abschließend sollen die Ergebnisse aller Werkzeuge und der Erzeugung dokumentiert werden.

Die Umsetzung dieses Anwendungsfalls und demzufolge die Schnittstelle mit der möglichen Lösung stellt ein *Batchfile* dar (siehe Anhang B). In diesem ist der Buildvorgang mit allen zugehörigen Schritten in einer definierten Syntax beschrieben. Ein solches Batchfile existiert bereits in der Entwicklungsumgebung der SMT und muss in die Lösung eingebunden werden. Batchfiles werden über die *Windows Shell* aufgerufen – eine mögliche Lösung muss also das Ausführen eines *Windows Shell Skriptes* unterstützen. Innerhalb des Skriptes wird zum Ablageort des Batchfiles navigiert woraufhin dieses ausgeführt wird.
Im Folgenden wird dieser Anwendungsfall abgekürzt als *Build* bezeichnet.

Anwendungsfall 2: Durchführung einer statischen Code-Analyse für ein/e Software-System/-Komponente

Eng hiermit verknüpft ist der nächste Anwendungsfall, der die Durchführung einer statischen Codeanalyse beschreibt. Diese umfasst die Evaluierung des Codes hinsichtlich bestehender Programmierrichtlinien sowie bestehender *Best-Practices*.
Hierfür wählt der Ausführende das betreffende Softwaresystem bzw. -komponente sowie deren gewünschte Revision aus. Die Analyse soll zeitgesteuert gestartet werden. Das Dateisystem wird nun aus der Versionsverwaltung mit den notwendigen Dateien gefüllt und es werden die Analysewerkzeuge ausgeführt. Abschließend sollen die Ergebnisse aller Analysewerkzeuge dokumentiert werden.
Zurzeit wird zur statischen Codeanalyse das Tool *PC-Lint*[2] eingesetzt – es wird also dessen Kompatibilität mit den möglichen Lösungswegen geprüft.
Im Folgenden wird dieser Anwendungsfall abgekürzt als *statische Codeanalyse* bezeichnet.

Anwendungsfall 3: Durchführung des Unit Tests für ein/e Software-System/-Komponente

Im Anschluss an die Erzeugung mit Codeanalyse werden Unit Tests durchgeführt. Hierbei werden Software-Einheiten (*Units*) getestet. Diese Tests werden auf einer dedizierten Hardware ausgeführt. Hierfür ist ein Unit-Test-Framework vorgesehen, das bereits benutzt wird und welches mit dem ausgewählten Werkzeug zusammen arbeiten soll.
Hierfür wählt der Ausführende das betreffende Softwaresystem oder die betreffende Systemkomponente sowie deren gewünschte Revision aus. Die Durchführung der Unit Tests soll zeitgesteuert gestartet werden. Das Dateisystem wird nun aus der Versionsverwaltung mit den notwendigen Dateien gefüllt und es wird die Erzeugung für die Unit Tests ausgeführt. Die Ergebnisse der Erzeugungswerkzeuge und der Erzeugung selbst sollen dokumentiert werden. Daraufhin werden die Unit Tests durchgeführt und deren Ergebnisse dokumentiert.

Das oben erwähnte *Unit-Test-Framework* wurde von der FORTecH Software GmbH für die SMT-Entwicklungsumgebung entwickelt. Es ist an die *XUnit Frameworks* angelehnt.
Es gibt zwei Möglichkeiten, dieses an potentielle Werkzeuge anzubinden:

Mittels Zugriff über die *command line utility:*

> *"The command line utility can be used to add unit testing to your build pipeline. Basically, it is able to run a selection of test cases in a console window. The diagnostic output is sent to the console. The rating of the whole test suite is returned as process*

[2] http://www.gimpel.com/html/pcl.htm (abgerufen am 24. August 2014)

> *result value and may be used to control the execution flow in a script."* (FORTeCH Software GmbH - UnitTest .NET Version 4.00 Software User Documentation, 2013-01-15, Revision A, Kap. 5.2, Z. 1 ff.)

Mittels eines *JUnit-compatible XML report:*

> *"Optionally, a Junit-compatible XML report can be created. This simplifies interaction with build servers like Hudson."* (FORTeCH Software GmbH – UnitTest .NET Version 4.00 Software User Documentation, 2013-01-15, Revision A, Kap. 5.2, Z. 6 f.)

Im Folgenden wird dieser Anwendungsfall als *Unit Test(s)* bezeichnet.

Anwendungsfall 4: Durchführung eines Systemtests

Der letzte Anwendungsfall beschreibt die Durchführung von Systemtests. Diese testen alle beteiligten Softwaresysteme im Rahmen eines Systems. Die notwendigen Softwaresysteme werden erzeugt und in eine bereitstehende Hardware geladen. Es wird das dynamische Verhalten der vom System zur Verfügung gestellten Schnittstellen überprüft. Die Ergebnisse der Systemtests sollen dokumentiert werden.
Hierfür wählt der Ausführende das betreffende Softwaresystem oder die betreffende Systemkomponente sowie deren gewünschte Revision aus. Die Durchführung der Systemtests soll zeitgesteuert gestartet werden. Das Dateisystem wird nun aus der Versionsverwaltung mit den notwendigen Dateien gefüllt und es wird die Erzeugung der ausgewählten Softwaresysteme für die Systemtests ausgeführt. Die Ergebnisse der Erzeugungswerkzeuge und der Erzeugung sollen dokumentiert werden. Es werden dann die Softwaresysteme in das zu testende System geladen und die Systeme zur Interaktion mit den Schnittstellen des zu testenden Systems führen die definierten Tests durch. Die Systemtests werden durchgeführt und auch deren Ergebnisse sollen dokumentiert werden.
Im Folgenden wird dieser Anwendungsfall als *Systemtest(s)* bezeichnet.

2.2 Funktionale und Nicht funktionale Anforderungen

Diese Anforderungen sowie die oben beschriebenen Anwendungsfälle bilden die SMT-Entwicklungsumgebung ab, an die eine mögliche Lösung angepasst sein muss:
Der Quellcode wird in der integrierten Entwicklungsumgebung (*IDE – Integrated Development Environment*) *Eclipse* geschrieben, zur statischen Codeanalyse wird *Lint* eingesetzt. Das Unit-Test-Framework ist eine Sonderentwicklung für SMT, die an die *XUnits-Frameworks* angelehnt ist. Zur Versionsverwaltung wird *Subversion* eingesetzt, als Bugtrackingtool *Jira*. Zudem wird noch das *Application Lifecycle Management Tool*

Polarion eingesetzt, das die Ergebnisse der CI-Umgebung entgegennehmen können sollte. Die Dokumentation erfolgt in Microsoft Office 2010.

Die folgenden Tabellen zeigen die Anforderungen aus dem Lastenheft sowie deren dortige Bezeichnung.

Tabelle 1: Funktionale Anforderungen

Funktionale Anforderungen	
Bezeichnung	**Anforderung**
L00021-23-1	**Zuordnung zu Jobs** Die Werkzeuge müssen fähig sein, Anwendungsfälle einem Job zuzuordnen. Teile von Anwendungsfällen, einzelne Anwendungsfälle oder mehrere Anwendungsfälle sollen als Job zusammengefasst werden.
L00021-23-2	**Anzahl möglicher Jobs** Die Werkzeuge sollen fähig sein, eine unbegrenzte Anzahl von Jobs zu definieren. In einem aktuellen Projekt der SMT gibt es gut 60 Softwarekomponenten und 10 Softwaresysteme. In feiner Granularität möge es somit jeweils 70 der ersten drei der oben genannten Anwendungsfälle geben. Wollte jeder der derzeit 10 Entwickler diese Anzahl von Anwendungsfällen zu Jobs zuordnen, sind 2000 Jobs notwendig. Ist die Vorgabe *unbegrenzte Anzahl von Jobs* nicht erreichbar, möge diese Größe ein Richtwert sein.
L00021-23-3	**Zeitgesteuerter Auslöser** Die Werkzeuge müssen fähig sein, Jobs zeitgesteuert zu starten. Eine Steuerung über Termine (auch regelmäßig wiederkehrende, beispielsweise täglich oder wöchentlich) ist notwendig
L00021-23-4	**Ereignisgesteuerter Auslöser** Die Werkzeuge müssen fähig sein, Jobs ereignisgesteuert zu starten. Ereignisse können sich aus anderen Systemen oder Werkzeugen selbst ergeben.
L00021-23-5	**Auslösung durch *Polarion*** Die Werkzeuge sollen dem *Application Lifecycle Management Tool Polarion* die Möglichkeit bieten, einen Job zu starten. Denkbar ist, dass mit dem Zustandswechsel eines *Work Items* ein Job gestartet wird.
L00021-23-6	**Auslösung durch *Subversion*** Die Werkzeuge sollten dem Versionsverwaltungssystem *Subversion* die Möglichkeit bieten, einen Job zu starten. Denkbar ist, dass mit dem Commit in das Repository ein Job gestartet wird.
L00021-23-7	**Auslösung durch *Eclipse*** Die Werkzeuge sollten der Entwicklungsumgebung *Eclipse* die Möglichkeit bieten, einen Job zu starten. Hier lassen sich eventuell die Möglichkeiten der Werkzeuge und die Möglichkeiten von *Eclipse* kombinieren. Für *Eclipse* steht eine Vielzahl von Plug-Ins zur Verfügung.
L00021-23-8	**Darstellung aktueller Job-Ergebnisse** Die Werkzeuge müssen fähig sein, aktuelle Job-Ergebnisse darzustellen.
L00021-23-9	**Darstellung vergangener Job-Ergebnisse** Die Werkzeuge müssen fähig sein, in der Vergangenheit erzeugte Job-Ergebnisse darzustellen.

L00021-23-10	**Dokumentation in einem Microsoft Office kompatiblen Format** Die Werkzeuge müssen fähig sein, Job-Ergebnisse in einem zu Microsoft Office 2010 kompatiblen Format zu dokumentieren. Microsoft Office 2010 ist der derzeit eingesetzte Standard bei SMT
L00021-23-11	**Dokumentation im PDF-Format** Die Werkzeuge sollten fähig sein, Job-Ergebnisse im PDF-Format zu dokumentieren
L00021-23-12	***Test Results* für *Polarion*** Die Werkzeuge sollten fähig sein, ein *Test Result* für *Polarion* zu liefern. Die Ergebnisse eines Jobs sollten derart formatierbar sein, dass sie in *Polarion* als Ergebnis eines *Test Runs* verwertet werden können. Hier lassen sich eventuell die Möglichkeiten der Werkzeuge und die Möglichkeiten von *Polarion* kombinieren. *Test Results* lassen sich in *Polarion* aus Excel-Tabellen extrahieren.
L00021-23-13	**Vorgänge in *Jira*** Die Werkzeuge müssen fähig sein, Vorgänge in *Jira* zu erzeugen. Im Falle eines fehlgeschlagenen Jobs soll ein *Jira*-Vorgang erzeugt werden. Hier lassen sich eventuell die Möglichkeiten der Werkzeuge und die Möglichkeiten von Jira kombinieren. Die *Jira*-Installation bei SMT bietet die Möglichkeit, Vorgänge aus E-Mails an issues@soering.com zu erzeugen. Zudem steht für *Jira* eine Vielzahl von AddOns zur Verfügung.

Tabelle 2: Nicht funktionale Anforderungen

Nicht funktionale Anforderungen	
Bezeichnung	**Anforderung**
L00021-24-1	**Einstellungsmöglichkeit im Webbrowser** Die Werkzeuge sollten Anwendern die Möglichkeit bieten, Einstellungen über einen Webbrowser vorzunehmen. Microsoft Internet Explorer 10 ist der derzeit eingesetzte Standard bei SMT. Mozilla Firefox wird bei SMT geduldet.
L00021-24-2	**Ergebnisabruf im Webbrowser** Die Werkzeuge sollten Anwendern die Möglichkeit bieten, Ergebnisse über einen Webbrowser abzurufen. Microsoft Internet Explorer 10 ist der derzeit eingesetzte Standard bei SMT. Mozilla Firefox wird bei SMT geduldet.
L00021-24-3	**Kompatibilität zu Windows Server 2008** Die Werkzeuge müssen auf einem zum Windows Server 2008 kompatiblen Server zu installieren sein. Windows Server 2008 ist der derzeit eingesetzte Standard bei SMT.
L00021-24-4	**Kompatibilität zu virtuellen Servern** Die Werkzeuge müssen auf einem virtuellen Server zu installieren sein. Der Einsatz von virtuellen Servern ist der derzeit eingesetzte Standard bei SMT.
L00021-24-5	**Immerwährendes Lizenzmodell** Die Werkzeuge müssen die Möglichkeit bieten, mit einem immerwährenden Lizenzmodell betrieben zu werden. Lizenzgebühren dürfen einmalig anfallen. Wiederkehrende Wartungsgebühren sind akzeptabel.

3 Methodik

Bei allen in Frage kommenden Lösungsmöglichkeiten wird geprüft, ob mit ihnen die geforderten Anwendungsfälle durchführbar sind und ob sie darüber hinaus die funktionalen und nicht funktionalen Anforderungen erfüllen. Auf dieser Grundlage werden Lösungsmöglichkeiten ausgesucht – hierbei soll nachvollziehbar sein, warum diese die Anforderungen aus dem Lastenheft am besten erfüllen. Mit Hilfe von Testinstallationen sollen die möglichen Werkzeuge anschließend eingehender bewertet werden.
Umgekehrt soll auch nachvollziehbar sein, warum andere Lösungsmöglichkeiten verworfen werden indem aufgezeigt wird, welche Anforderungen sich gar nicht oder nur schlecht umsetzen lassen.

Die Bewertung der möglichen Lösungen erfolgt hierbei in drei Schritten, die aufeinander aufbauen. Im ersten Schritt wird für jeden Punkt im Lastenheft geprüft, ob und wie die zu bewertende Lösung die in diesem Punkt geforderte Anforderung umsetzt. Diese erste Bewertung erfolgt auf Grundlage einer detaillierten Recherche, sodass sie durch Quellen belegbar ist. Es handelt sich bei diesem ersten Schritt folglich um eine *qualitative* Beurteilung.
Im Gegensatz hierzu soll im zweiten Schritt eine *quantitative* Bewertung erfolgen. Es werden dabei die ersten ermittelten Ergebnisse quantifiziert, sodass alle möglichen Lösungen untereinander direkt vergleichbar werden.
Der dritte Schritt sieht vor, ein *Test Setup* aus den Lösungen zu erstellen, die sich im zweiten Schritt als aussichtsreich erwiesen haben.

4 Lösungsansatz

Ziel ist es, ein passendes Werkzeug zu finden, mit dem Kontinuierliche Integration nach den im vorangegangenen Kapitel beschriebenen Anforderungen möglich ist. Im ersten Anlauf bedeutet dies, die vier Anwendungsfälle *statische Codeanalyse, Build, Unit Tests und Systemtests*, die heute von jedem Entwickler separat und größtenteils auf ihren Arbeitsplatzrechnern durchgeführt werden, zu automatisieren. Diese Automatisierung kann konkret ein zeitgesteuertes (L00021-23-3) oder ein ereignisgesteuertes Auslösen des Buildvorgangs und der Tests bedeuten (L00021-23-4). Ereignisse können etwa ein Zustandswechsel in *Polarion* (L00021-23-5) oder ein *Commit* in das *Subversion Repository* (L00021-23-6) sein.
Ein weiteres wesentliches Merkmal der Kontinuierlichen Integration besteht darin, die Ergebnisse dieser vier Anwendungsfälle zu dokumentieren und bestenfalls so aufzubereiten, dass diese leicht und zentral zugänglich sind und sich beispielsweise über einen gewissen Zeitraum vergleichen lassen. Dies wird in den funktionalen Anforderungen L00021-23-8 und L00021-23-9 ebenfalls gefordert. Idealerweise werden die Ergebnisse in übersichtlicher Form in einem Webbrowser präsentiert.

Ein möglicher Lösungsansatz wäre es, verschiedene Skripte für diese Anforderungen zu schreiben. Diese ließen sich individuell an die vorhandene Entwicklungsumgebung anpassen. Problematisch wären dabei allerdings jene Anforderungen, die die Interaktion mit vielen verschiedenen Systemen wie *Eclipse, Jira, Polarion, Subversion* und E-Mail betreffen. Zudem könnten Updates an solchen Systemen eine Anpassung der Skripte erforderlich machen. Das entstehende Sammelsurium an unterschiedlichen Skripten wäre darüber hinaus recht unübersichtlich, fehleranfällig und wahrscheinlich schwer zu warten.

Eine weitere Möglichkeit stellen daher die sogenannten *Build-Server* oder seltener auch *Continuous Integration Server* dar. Hierbei handelt es sich um speziell auf Kontinuierliche Integration zugeschnittene Software, die sich beispielsweise aus einem Versionsverwaltungssystem die nötigen Komponenten lädt, um automatisch zu builden. In der Regel stehen darüber hinaus Plug-Ins zur Verfügung, die diese Software um zusätzliche Funktionalitäten erweitern – etwa, um *Jira* zu integrieren oder die Ergebnisse von Tests aufzubereiten und zu präsentieren – und sie somit an eine gegebene Entwicklungsumgebung anpassbar zu machen. Das bedeutet, die Anwendungsfälle 1 bis 3 ließen sich grundsätzlich mit Hilfe eines *Build-Servers* umsetzen. Anwendungsfall 4 sieht ein „skriptfähiges System zur Interaktion mit den

Schnittstellen des zu testenden Systems“ vor – mit einem Plug-In, das die Verarbeitung von Skripten erlaubt, ließe sich auch dieser realisieren.
Als zentraler Ausführungsort für alle mit dem Buildvorgang zusammenhängenden Aktivitäten böte ein *Build-Server* eine deutlich strukturiertere Lösungsmöglichkeit als eine skriptbasierte Lösung, die auf den lokalen Arbeitsstationen der Entwickler ausgeführt würde. Möglichen Problemen beim Austausch einzelner Arbeitsstationen würde ebenfalls vorgebeugt: Die Serverhardware und Software bleibt nach Einrichtung über einen langen oder möglicherweise gar den gesamten Zeitraum des Software-Lebenszyklus unverändert, sodass ein definierter Stand vorliegt. Auch für die Reproduzierbarkeit von Testergebnissen, die im Zuge des Buildvorganges entstehen, ist ein zentraler Ausführungsort somit von Vorteil.

Die Lösung mittels *Build-Server* genügt somit den Anforderungen und lässt sich zudem einfacher und strukturierter umsetzen. Aus diesem Grund soll dem Lösungsansatz an dieser Stelle der Vorzug gegeben werden. Das bedeutet konkret, aus der Vielzahl an verfügbaren Produkten jenes zu finden, welches alle Anforderungen erfüllt und mit dem sich diese am besten umsetzen lassen. Weitere nicht funktionale Anforderungen können bei dieser Auswahl ebenfalls über die Anforderungen des Lastenheftes hinaus eine Rolle spielen – jedoch nur, wenn all diese Anforderungen erfüllt werden können. Zu diesen weitergehenden Anforderungen zählt in erster Linie die Handhabung: Installation, Konfiguration, Erscheinungsbild und Übersichtlichkeit der graphischen Benutzeroberfläche sowie weitere Aspekte der *Usability*.

5 Qualitative Bewertung

5.1 Vorbetrachtungen

Nachdem ein Lösungsansatz gewählt wurde, der die Auswahl eines geeigneten *Build-Servers* vorsieht, sollen an dieser Stelle verschiedene *Build-Server* vorgestellt und anhand der beschriebenen Kriterien bewertet werden. Diese Auswahl bezieht dabei nur solche Lösungen ein, die grundsätzlich in Frage kommen. Die Auswertung der verschiedenen *Build-Server* bezieht sich auf die Anwendungsfälle sowie die funktionalen und nicht funktionalen Anforderungen aus dem Lastenheft.
Die Realisierbarkeit der Anwendungsfälle muss gesondert bewertet werden, da hier – anders als bei den konkreten Anforderungen – mehrere Faktoren hineinspielen, die sich unter Umständen noch nicht exakt benennen lassen. Dadurch, dass die Anwendungsfälle abstrakter sind, hat deren Beurteilung eher den Charakter eines Ausblicks, der durch eine Testinstallation verifiziert werden muss.
Im Anschluss an dieses Kapitel soll ein Bewertungssystem eingeführt werden, mit dessen Hilfe sich die hier vorgestellten *Build-Server* quantifizierbar miteinander vergleichen lassen.

5.2 Hudson

Hudson ist ein kostenloser, quelloffener CI-Server. Es folgt eine tabellarische Aufstellung darüber, welche Anwendungsfälle und Anforderungen sich mit Hudson umsetzen ließen. Dabei handelt es sich um Rechercheergebnisse, die noch unvollständig sein können – eine lückenlose Aufstellung über die Erfüllung bzw. Nichterfüllung aller Anforderungen lässt sich erst nach einer Testinstallation machen. Im Anschluss an diese Tabelle soll eine erste Abschätzung stehen, ob Hudson mit Hilfe einer Testinstallation näher bewertet werden sollte.

Tabelle 3: Realisierbarkeit der Anwendungsfälle bei Hudson

Anwendungsfälle	
Build	Der Aufruf von Batchfiles ist von Hudson vorgesehen [97].
Statische Codeanalyse	Das Tool Lint lässt sich mit Hilfe eines Plug-Ins einbinden [109].
Unit Tests	Hudson sieht standardmäßig die Einbindung eines JUnit-Test-Frameworks vor [102]. Das bei SMT eingesetzte Unit-Test-Framework liefert einen zu JUnit kompatiblen XML-Output.

Systemtests	Dieser Anwendungsfall muss in einer Testinstallation überprüft werden.

Tabelle 4: Realisierbarkeit der funktionalen Anforderungen bei Hudson

Funktionale Anforderungen	
L00021-23-1	Hudson ist fähig, Anwendungsfälle einem Job zuzuordnen [1].
L00021-23-2	Hudson ist fähig, eine theoretisch unbegrenzte Anzahl von Jobs zu definieren und zu verwalten.
L00021-23-3	Hudson ist fähig, Jobs zeitgesteuert zu starten [2].
L00021-23-4	Hudson ist fähig, Jobs ereignisgesteuert zu starten [3].
L00021-23-5	Es ist sowohl ein Polarion Plug-In für Hudson [4] als auch ein Hudson Plug-In für Polarion [5] verfügbar. Ob die Anforderung hiermit umsetzbar ist, müsste mit Hilfe einer Testinstallation überprüft werden.
L00021-23-6	Hudson sieht bereits in der Basisdistribution den Einsatz von Subversion vor [6].
L00021-23-7	Eclipse und Hudson lassen sich über ein Plug-In für Eclipse verknüpfen [7].
L00021-23-8	Hudson sieht vor, die aktuellen Job-Ergebnisse in einer Weboberfläche aufbereitet zu präsentieren [8].
L00021-23-9	Hudson sieht vor, in der Vergangenheit erzeugte Job-Ergebnisse in einer Weboberfläche aufbereitet zu präsentieren [9].
L00021-23-10	Nach Recherche lässt sich nicht feststellen, ob die Dokumentation von Job-Ergebnissen in einem zu Microsoft Office 2010 kompatiblen Format vorgesehen ist.
L00021-23-11	Nach Recherche lässt sich nicht feststellen, ob die Dokumentation von Job-Ergebnissen in einem PDF-Dateiformat vorgesehen ist.
L00021-23-12	Hier wäre eine Testinstallation abzuwarten, um zu überprüfen, ob das Polarion Plug-In [10] diese Anforderung eventuell umsetzen kann. Eine weitere Möglichkeit wäre der Umweg über eine Excel-Tabelle – hierfür müsste allerdings aus den Test-Results erst eine solche generiert werden.
L00021-23-13	Jira und Hudson lassen sich sowohl über ein Plug-In für Jira [11] als auch über ein Plug-In für Hudson [12] verknüpfen.

Tabelle 5: Realisierbarkeit der nicht funktionalen Anforderungen bei Hudson

Nicht funktionale Anforderungen	
L00021-24-1	Hudson sieht vor, dass Anwender über ein Webinterface Einstellungen vornehmen [13].
L00021-24-2	Hudson sieht vor, dass Anwender Ergebnisse über ein Webinterface abrufen [14].
L00021-24-3	Hudson ist kompatibel zu Windows Betriebssystemen [15].
L00021-24-4	Hudson ist kompatibel zu Windows Betriebssystemen [16].
L00021-24-5	Hudson steht unter der *Eclipse EPL Licence* [17]. Diese erlaubt die kostenlose Benutzung von Hudson.

Schon vor einer Testinstallation wird erkennbar, dass Hudson den Großteil der Anforderungen erfüllen kann. Der *Build-Server* bringt bereits in der Basisdistribution eine umfangreiche Funktionalität mit, die durch das Konzept der Plug-Ins noch erweiterbar ist. Die kostenlose Verfügbarkeit ist ein weiteres attraktives Merkmal.

5.3 Jenkins

Jenkins ist ebenfalls kostenlos und quelloffen, da dieser CI-Server unter MIT-Lizenz [18] steht.
2011 ist Jenkins aus Hudson hervorgegangen. Ursprünglich wurde Hudson bei Sun Microsystems von Kohsuke Kawaguchi entwickelt. Als dieses Unternehmen von Oracle übernommen wurde, entschieden sich einige Entwickler – darunter auch Kawaguchi – , das Projekt Hudson unter dem Namen Jenkins außerhalb von Oracle fortzuführen. Parallel dazu wird heute jedoch Hudson von Oracle weiterentwickelt.
Aus diesem Grund gibt es starke Ähnlichkeiten zwischen Hudson und Jenkins. Dadurch, dass jedoch der ursprüngliche Entwickler an Jenkins weiterarbeitet, ist dieses heute sehr viel beliebter und weiter verbreitet als Hudson. Das bedeutet, dass auch die Anzahl der zur Verfügung stehenden Plug-Ins bei Jenkins größer ist.

Tabelle 6: Realisierbarkeit der Anwendungsfälle bei Jenkins

Anwendungsfälle	
Build	Der Aufruf von Batchfiles ist von Jenkins vorgesehen [98].
Statische Codeanalyse	Das Tool Lint lässt sich mit Hilfe eines Plug-Ins einbinden [110].
Unit Tests	Jenkins sieht standardmäßig die Einbindung eines JUnit-Test Frameworks vor [103]. Das bei SMT eingesetzte Unit-Test Framework liefert einen zu JUnit kompatiblen XML-Output.
Systemtests	Dieser Anwendungsfall muss in einer Testinstallation überprüft werden.

Tabelle 7: Realisierbarkeit der funktionalen Anforderungen bei Jenkins

Funktionale Anforderungen	
L00021-23-1	Jenkins ist fähig, Anwendungsfälle einem Job zuzuordnen [19].
L00021-23-2	Es gibt keine theoretischen Beschränkungen für die Anzahl an möglichen Jobs.
L00021-23-3	Jenkins ist fähig, Jobs zeitgesteuert zu starten [20].
L00021-23-4	Jenkins ist fähig, Jobs ereignisgesteuert zu starten [21].
L00021-23-5	Polarion lässt sich über ein Plug-In [22] einbinden. Ob die Anforderung hiermit umsetzbar ist, müsste mit Hilfe einer Testinstallation überprüft werden.
L00021-23-6	Jenkins sieht bereits in der Basisdistribution den Einsatz von Subversion vor [23].
L00021-23-7	Eclipse und Jenkins lassen sich über ein Plug-In für Eclipse verknüpfen [24].
L00021-23-8	Jenkins sieht vor, die aktuellen Job-Ergebnisse in einer Weboberfläche aufbereitet zu präsentieren [25].
L00021-23-9	Jenkins sieht vor, in der Vergangenheit erzeugte Job-Ergebnisse in einer Weboberfläche aufbereitet zu präsentieren [26].
L00021-23-10	Nach Recherche lässt sich nicht feststellen, ob die Dokumentation von Job-Ergebnissen in einem zu Microsoft Office 2010 kompatiblen Format vorgesehen ist.
L00021-23-11	Nach Recherche lässt sich nicht feststellen, ob die Dokumentation von Job-Ergebnissen in einem PDF-Dateiformat vorgesehen ist.
L00021-23-12	Hier wäre eine Testinstallation abzuwarten, um zu überprüfen, ob das Polarion Plug-In [27] dies eventuell leisten kann. Eine weitere Möglichkeit wäre der Umweg über eine Excel-Tabelle – hierfür müsste allerdings aus den Test-Results erst eine solche generiert werden.

L00021-23-13	Jira und Jenkins lassen sich sowohl über ein Plug-In für Jira [28] als auch über ein Plug-In für Jenkins [29] verknüpfen.

Tabelle 8: Realisierbarkeit der nicht funktionalen Anforderungen bei Jenkins

Nicht funktionale Anforderungen	
L00021-24-1	Jenkins sieht vor, dass Anwender über ein Webinterface Einstellungen vornehmen [30].
L00021-24-2	Jenkins sieht vor, dass Anwender Ergebnisse über ein Webinterface abrufen [31].
L00021-24-3	Jenkins ist kompatibel zu Windows Betriebssystemen [32].
L00021-24-4	Jenkins ist kompatibel zu Windows Betriebssystemen [33].
L00021-24-5	Jenkins steht unter der MIT-Lizenz. Diese erlaubt die kostenlose Benutzung von Jenkins [34].

Hudson vs. Jenkins

Hudson und Jenkins haben aufgrund ihrer gemeinsamen Wurzeln eine große Ähnlichkeit; die grundlegende Funktionsweise ist dieselbe. Dennoch ist Jenkins heute viel weiter verbreitet als Hudson. Der Grund hierfür mag sein, dass Jenkins vom ursprünglichen Team um Kawaguchi weiter entwickelt wird.
Für Jenkins sind derzeit über 600 Plug-Ins erhältlich [35], während in der *Hudson Plugin Central* [36] derzeit knapp 400 Plug-Ins gelistet sind. Da Plug-Ins auch von Anwendern selber geschrieben werden können, verwundert es nicht, dass dort, wo mehr Nutzer sind, die größere Auswahl an Plug-Ins anzutreffen ist.
Ein weiterer Unterschied betrifft das Release-Modell: Während von Hudson seltener neue Versionen erscheinen, gibt es bei Jenkins beinahe wöchentlich einen sogenannten *latest and greatest*-Release. Es gilt also abzuwägen: Kurze Release-Zyklen mit schnelleren Bugfixes und neuen Features gegen Stabilität und Long Term Support.
Allerdings bietet Jenkins neben dem wöchentlichen Release auch einen Long Term Support an: Hierfür wird alle zwölf Wochen eine aktuelle und stabile Version ausgewählt und im *Jenkins Long Term Support release* [37] angeboten, sodass man bei Jenkins die Wahl zwischen häufigen oder stabileren Releases hat.
Ungeachtet der Unterschiede bieten sowohl Hudson als auch Jenkins eine den Anforderungen entsprechende Funktionalität.

5.4 Bamboo

Bamboo ist ein kommerzieller CI-Server von Atlassian, demselben Hersteller wie von Jira. Dementsprechend ist eine Anbindung an das *Issuetracking*, für das bei SMT *Jira* verwendet wird, sehr einfach. So ist beispielsweise standardmäßig bereits die Möglichkeit vorgesehen, durch fehlgeschlagene Buildvorgänge Tickets erstellen zu lassen.

Tabelle 9: Realisierbarkeit der Anwendungsfälle bei Bamboo

Anwendungsfälle	
Build	Der Aufruf von Batchfiles ist von Bamboo vorgesehen [99].
Statische Codeanalyse	Dieser Anwendungsfall muss in einer Testinstallation überprüft werden.
Unit Tests	Bamboo sieht standardmäßig die Einbindung eines JUnit-Test Frameworks vor [104]. Das bei SMT eingesetzte Unit-Test Framework liefert einen zu JUnit kompatiblen XML-Output.
Systemtests	Dieser Anwendungsfall muss in einer Testinstallation überprüft werden.

Tabelle 10: Realisierbarkeit der funktionalen Anforderungen bei Bamboo

Funktionale Anforderungen	
L00021-23-1	Bamboo ist fähig, Anwendungsfälle einem Job zuzuordnen [38].
L00021-23-2	Bamboo bietet die Möglichkeit, eine unbegrenzte Anzahl von Jobs zu definieren [39]. Siehe hierzu auch Punkt 24-5 in Tabelle 11.
L00021-23-3	Bamboo ist fähig, Jobs zeitgesteuert zu starten [40].
L00021-23-4	Bamboo ist fähig, Jobs ereignisgesteuert zu starten [41].
L00021-23-5	Bamboo besitzt standardmäßig keine Schnittstelle zu Polarion. Polarion lässt sich nur umständlich in eine Bamboo-Umgebung integrieren. Hierfür ist eine weitere kostenpflichtige Software notwendig, die allerdings weit mehr beherrscht als *nur* die Verknüpfung von Bamboo mit Polarion [42]. Diese sperrige Lösung ist daher nicht sonderlich praktikabel – auch, weil weitere Kosten für eine Software entstehen würden, die weit mehr beherrscht als eigentlich benötigt.
L00021-23-6	Bamboo sieht den Einsatz von Subversion vor [43].
L00021-23-7	Um Bamboo-Jobs aus Eclipse heraus zu starten, wird ein Plug-In für Eclipse benötigt [45].
L00021-23-8	Bamboo ist fähig, aktuelle Job-Ergebnisse darzustellen.
L00021-23-9	Bamboo ist fähig, in der Vergangenheit erzeugte Job-Ergebnisse darzustellen.
L00021-23-10	Nach Recherche lässt sich nicht feststellen, ob die Dokumentation von Job-Ergebnissen in einem zu Microsoft Office 2010 kompatiblen Format vorgesehen ist.
L00021-23-11	Nach Recherche lässt sich nicht feststellen, ob die Dokumentation von Job-Ergebnissen in einem PDF-Dateiformat vorgesehen ist.
L00021-23-12	Bamboo besitzt standardmäßig keine Schnittstelle zu Polarion. Polarion lässt sich nur umständlich in eine Bamboo-Umgebung integrieren. Hierfür ist eine weitere kostenpflichtige Software notwendig, die allerdings weit mehr beherrscht als *nur* die Verknüpfung von Bamboo mit Polarion [46]. Diese sperrige Lösung ist daher nicht sonderlich praktikabel - auch, weil weitere Kosten für eine Software entstehen würden, die weit mehr beherrscht als eigentlich benötigt. Eine weitere Möglichkeit wäre der Umweg über eine Excel-Tabelle – hierfür müsste allerdings aus den Test-Results erst eine solche generiert werden.
L00021-23-13	Bamboo ist in der Lage, Vorgänge – beispielsweise im Falle eines fehlgeschlagenen Jobs – in Jira zu erzeugen [47].

Tabelle 11: Realisierbarkeit der nicht funktionalen Anforderungen bei Bamboo

Nicht funktionale Anforderungen	
L00021-24-1	Bamboo sieht vor, dass Anwender über ein Webinterface Einstellungen vornehmen

	[48].
L00021-24-2	Bamboo sieht vor, dass Anwender Ergebnisse über ein Webinterface abrufen [49].
L00021-24-3	Bamboo ist kompatibel zu Windows Betriebssystemen [50].
L00021-24-4	Bamboo ist kompatibel zu Windows Betriebssystemen [51].
L00021-24-5	Das Bamboo-Lizenzmodell [52] sieht unterschiedliche Lizenzkosten vor, die von drei Faktoren abhängen: - Installation auf eigenem Server oder in der Cloud - Anzahl der möglichen Jobs - Anzahl der *Agents* Die cloudbasierte Lösung kommt nicht in Frage. Bei Installation auf einem eigenen Server entstehen einmalig Lizenzkosten, die davon abhängen, wie viele mögliche Jobs und wie viele Agents benötigt werden. Je mehr Agents zur Verfügung stehen, desto mehr Prozesse – desselben oder auch unterschiedlicher Buildvorgänge – können parallel ablaufen.

Tabelle 11a: Einmalig anfallende Lizenzkosten in Abhängigkeit der Anzahl möglicher Jobs und Build-Agents bei Installation auf eigenem Server

Anzahl der möglichen Jobs	Anzahl der Build Agents	Einmalige Lizenzkosten in US-Dollar
10	Unbegrenzt viele lokale / 0 Remote	10
Unbegrenzt	1 Remote	800
Unbegrenzt	5 Remote	2200
Unbegrenzt	10 Remote	4000
Unbegrenzt	25 Remote	8000
Unbegrenzt	100 Remote	16000

Bamboo deckt also einen Großteil der Anforderungen bereits standardmäßig ab, was auch diesen *Build-Server* interessant macht

5.5 Microsoft Team Foundation Server (TFS)

TFS ist eher eine Softwareentwicklungsplattform als *nur* ein *Build-Server*. So übernimmt TFS über die Automatisierung des Build- und Testvorganges hinaus zusätzlich die Aufgabe eines *source code management systems*, eines *issue trackers* sowie weiterer organisatorischer Unterstützungen. Dies sind allerdings Anwendungen, die im SMT-Entwicklungsprozess bereits von anderen Tools abgedeckt werden, sodass diese Features nicht benötigt würden. TFS ist sehr stark auf die Verwendung von Microsoft Visual Studio als Entwicklungsumgebung ausgelegt, es gibt jedoch ein Plug-In für Eclipse.
Dadurch, dass TFS nicht primär als *Build-Server* gedacht ist, ist die Umsetzung des Continuous Integration Gedanken nicht sofort *out of the box* möglich – jedenfalls nicht außerhalb einer komplett auf Microsoft ausgerichteten Entwicklungsumgebung.[3]

[3] Siehe hierzu auch *How to: Create an Automated Build and Deployment Solution with Team Foundation Server Team Build*: http://msdn.microsoft.com/en-us/library/ff650529.aspx (abgerufen am 25. August 2014)

Tabelle 12: Realisierbarkeit der Anwendungsfälle beim Team Foundation Server

Anwendungsfälle	
Build	Da Team Foundation Server auf die Zusammenarbeit mit Visual Studio ausgelegt ist, sind Batchfiles erst einmal nicht vorgesehen. Möglicherweise ließen sich Batchfiles über die *Buildplattform MSBuild* aufrufen [100].
Statische Codeanalyse	Es gibt die Möglichkeit der sogenannten *Custom Code Analysis Check-in Policies for Managed Code* [53]. Ob sich Lint direkt integrieren lässt, kann ohne Testinstallation nicht festgestellt werden.
Unit Tests	Das Plug-In *Microsoft Visual Studio Team Foundation Server 2013 Build Extension* [105] erfüllt die Aufgabe der Integration von JUnit Ergebnissen. Das bei SMT eingesetzte Unit-Test Framework liefert einen zu JUnit kompatiblen XML-Output.
Systemtests	Dieser Anwendungsfall muss in einer Testinstallation überprüft werden.

Tabelle 13: Realisierbarkeit der funktionalen Anforderungen beim Team Foundation Server

Funktionale Anforderungen	
L00021-23-1	Team Foundation Server ist fähig, Anwendungsfälle einem Job zuzuordnen [86].
L00021-23-2	Es gibt keine theoretischen Beschränkungen für die Anzahl an möglichen Jobs.
L00021-23-3	Team Foundation Server ist fähig, Jobs zeitgesteuert zu starten [87].
L00021-23-4	Team Foundation Server ist fähig, Jobs ereignisgesteuert zu starten [88].
L00021-23-5	Die Verknüpfung mit Polarion ist weder direkt durch Team Foundation Server noch durch ein Plug-In vorgesehen.
L00021-23-6	Die Verknüpfung mit Subversion ist mit einem Plug-In möglich [89].
L00021-23-7	Die Verknüpfung mit Eclipse ist mit einem Plug-In möglich [90].
L00021-23-8	Team Foundation Server ist fähig, aktuelle Job-Ergebnisse darzustellen [91].
L00021-23-9	Team Foundation Server ist fähig, in der Vergangenheit erzeugte Job-Ergebnisse darzustellen [91].
L00021-23-10	Nach Recherche lässt sich nicht feststellen, ob die Dokumentation von Job-Ergebnissen in einem zu Microsoft Office 2010 kompatiblen Format vorgesehen ist.
L00021-23-11	Nach Recherche lässt sich nicht feststellen, ob die Dokumentation von Job-Ergebnissen in einem PDF-Dateiformat vorgesehen ist.
L00021-23-12	Die Verknüpfung mit Polarion ist weder direkt durch Team Foundation Server noch durch ein Plug-In vorgesehen.
L00021-23-13	Für die Verknüpfung zwischen Team Foundation Server und Jira steht ein kostenpflichtiges Plug-In zur Verfügung [92]. Alternativ könnte auch im Falle eines fehlgeschlagenen Builds ein *build notification email alert* [93] aus Team Foundation Server heraus an issues@soering.com gesendet werden, sodass hierdurch in Jira ein Vorgang erzeugt würde.

Tabelle 14: Realisierbarkeit der nicht funktionalen Anforderungen beim Team Foundation Server

Nicht funktionale Anforderungen	
L00021-24-1	Hierfür wird *Team System Web Access* [94] benötigt. Ob hiermit allerdings eine Konfiguration möglich ist, kann noch nicht beurteilt werden.
L00021-24-2	Hierfür wird *Team System Web Access* [94] benötigt.

L00021-24-3	Team Foundation Server ist in den Versionen TFS 2010 und TFS 2012 (mit Service Pack 2) mit Windows Server 2008 kompatibel [95].
L00021-24-4	Team Foundation Server ist in den Versionen TFS 2010 und TFS 2012 (mit Service Pack 2) mit Windows Server 2008 kompatibel [95].
L00021-24-5	Team Foundation Server muss einmalig für 636,00 € erworben werden [96].

Nach der Recherche bleibt unklar, wie die Gesamtkonfiguration mit dem Team Foundation Server aussehen würde. So ist dieser CI-Server beispielsweise hauptsächlich auf die Benutzung von Microsoft Visual Studio als integrierte Entwicklungsumgebung ausgelegt. Und auch weitere Microsoft Software muss – je nach Anwendungsszenario – hinzugefügt werden; etwa *Team System Web Access* für den Zugriff vom Browser aus. Diese Unwägbarkeiten lassen von einer näheren Begutachtung absehen.

5.6 Cruise Control

Cruise Control – erstmals veröffentlicht im März 2001 [54] – war der erste verfügbare CI-Server. Da der letzte Release allerdings schon vier Jahre zurück liegt [55], kann Cruise Control möglicherweise, was Funktionalität und Kompatibilität zu aktuellen Versionen von anderen Programmen betrifft, nicht mehr mit moderneren CI-Servern mithalten. So erfolgt beispielsweise die gesamte Konfiguration über eine XML-Datei und nicht, wie bei vielen anderen CI-Servern üblich, über ein Web-Interface.

Tabelle 15: Realisierbarkeit der Anwendungsfälle bei Cruise Control

Anwendungsfälle	
Build	Der Aufruf von Batchfiles ist von Cruise Control vorgesehen [101].
Statische Codeanalyse	Es ist nicht klar, ob Cruise Control die Verwendung von Lint zulässt.
Unit Tests	Cruise Control sieht standardmäßig die Einbindung eines JUnit-Test Frameworks vor [106]. Das bei SMT eingesetzte Unit-Test-Framework liefert einen zu JUnit kompatiblen XML-Output.
Systemtests	Dieser Anwendungsfall muss in einer Testinstallation überprüft werden.

Tabelle 16: Realisierbarkeit der funktionalen Anforderungen bei Cruise Control

Funktionale Anforderungen	
L00021-23-1	Cruise Control ist fähig, Anwendungsfälle einem Job zuzuordnen [56].
L00021-23-2	Nach Recherche lässt sich keine Limitierung der Anzahl möglicher Jobs feststellen.
L00021-23-3	Cruise Control ist fähig, Jobs zeitgesteuert zu starten [57].
L00021-23-4	Cruise Control ist fähig, Jobs ereignisgesteuert zu starten [58].
L00021-23-5	Die Verknüpfung mit Polarion ist weder direkt durch Cruise Control noch durch ein Plug-In vorgesehen.
L00021-23-6	Cruise Control sieht die Anbindung an Subversion standardmäßig vor [59].
L00021-23-7	Es gibt ein Plug-In für Eclipse [60], welches jedoch diese Anforderung nicht erfüllt und auch nicht im offiziellen Eclipse Marketplace gelistet ist.
L00021-23-8	Cruise Control sieht vor, aktuelle Job-Ergebnisse in einer Weboberfläche

	darzustellen [61].
L00021-23-9	Cruise Control sieht vor, in der Vergangenheit erzeugte Job-Ergebnisse in einer Weboberfläche darzustellen [62].
L00021-23-10	Nach Recherche lässt sich nicht feststellen, ob die Dokumentation von Job-Ergebnissen in einem zu Microsoft Office 2010 kompatiblen Format vorgesehen ist.
L00021-23-11	Nach Recherche lässt sich nicht feststellen, ob die Dokumentation von Job-Ergebnissen in einem PDF-Dateiformat vorgesehen ist.
L00021-23-12	Da kein Plug-In verfügbar ist, bliebe nur der Weg über eine Excel-Tabelle, die allerdings auch erst aus den Test-Results erstellt werden müsste.
L00021-23-13	Es ist weder für Jira noch für Cruise Control [63] ein Plug-In verfügbar. Da Cruise Control jedoch in der Lage ist, E-Mail Benachrichtigungen zu versenden, könnte Jira aus diesen E-Mails Vorgänge erzeugen.

Tabelle 17: Realisierbarkeit der nicht funktionalen Anforderungen bei Cruise Control

Nicht funktionale Anforderungen	
L00021-24-1	Cruise Control sieht nicht vor, dass Anwender Einstellungen über ein Webinterface vornehmen. Hierfür ist das XML-File vorgesehen [64].
L00021-24-2	Cruise Control sieht vor, dass Anwender Ergebnisse über ein Webinterface abrufen [65].
L00021-24-3	Cruise Control lässt sich systemübergreifend installieren [66].
L00021-24-4	Cruise Control lässt sich systemübergreifend installieren [67].
L00021-24-5	Cruise Control ist frei verfügbar [68] .

Als erster etablierter CI-Server erfüllt Cruise Control natürlich die wesentlichen Anforderungen an einen solchen. Dennoch bieten andere Lösungen mittlerweile mehr. Die Entwicklung von Cruise Control scheint seit Jahren zu stagnieren. Die offizielle Webseite wird nicht gepflegt, so ist beispielsweise die Online-Dokumentation nicht mehr zu erreichen.[4]

5.7 Team City

Team City ist ein weiterer kommerzieller CI-Server, bei dem jedoch eine kostenlose Benutzung möglich ist. Hierdurch ergeben sich Einschränkungen wie etwa die Anzahl der *Build Configurations* oder der *Build Agents*. Team City ist jedoch nicht standardmäßig – wie alle bisher vorgestellten CI-Server – auf die Programmiersprache C++ ausgelegt. Hierfür ist ein Plug-In erforderlich [107].

Tabelle 18: Realisierbarkeit der Anwendungsfälle bei Team City

Anwendungsfälle	
Build	Der Aufruf von Batchfiles ist von Team City vorgesehen [69].
Statische Codeanalyse	Es ist kein Plug-In für Lint vorhanden [70]. Eine Möglichkeit, die Ausgabe von Lint zu parsen, müsste mit Hilfe einer Testinstallation geprüft

[4] http://confluence.public.thoughtworks.org/display/CC/Home (abgerufen am 16. September 2014)

	werden.
Unit Tests	Team City sieht standardmäßig die Einbindung eines JUnit-Test-Frameworks vor [108]. Das bei SMT eingesetzte Unit-Test Framework liefert einen zu JUnit kompatiblen XML-Output.
Systemtests	Dieser Anwendungsfall muss in einer Testinstallation überprüft werden.

Tabelle 19: Realisierbarkeit der funktionalen Anforderungen bei Team City

Funktionale Anforderungen	
L00021-23-1	Team City ist fähig, Anwendungsfälle einem Job zuzuordnen [71].
L00021-23-2	Team City bietet die Möglichkeit, eine unbegrenzte Anzahl an Jobs zu definieren. Siehe hierzu auch Punkt 24-5 in Tabelle 20.
L00021-23-3	Team City ist fähig, Jobs zeitgesteuert zu starten [72].
L00021-23-4	Team City ist fähig, Jobs ereignisgesteuert zu starten [73].
L00021-23-5	Die Verknüpfung mit Polarion ist weder direkt durch Cruise Control noch durch ein Plug-In vorgesehen.
L00021-23-6	Team City sieht eine Anbindung an Subversion standardmäßig vor [74].
L00021-23-7	Mit Hilfe des Eclipse-Plug-Ins [75] lässt sich ein sogenannter *Remote Run* ausführen.
L00021-23-8	Team City sieht vor, aktuelle Job-Ergebnisse in einer Weboberfläche darzustellen [76].
L00021-23-9	Team City sieht vor, in der Vergangenheit erzeugte Job-Ergebnisse in einer Weboberfläche darzustellen [77].
L00021-23-10	Nach Recherche lässt sich nicht feststellen, ob die Dokumentation von Job-Ergebnissen in einem zu Microsoft Office 2010 kompatiblen Format vorgesehen ist.
L00021-23-11	Nach Recherche lässt sich nicht feststellen, ob die Dokumentation von Job-Ergebnissen in einem PDF-Dateiformat vorgesehen ist.
L00021-23-12	Da kein Plug-In verfügbar ist, bliebe nur der Weg über eine Excel-Tabelle, die allerdings auch erst aus den Test-Results erstellt werden müsste.
L00021-23-13	Mit Hilfe des *TeamCity JIRA Reporter Plug-Ins* [78] lassen sich Vorgänge in Jira verändern, verschieben und erzeugen.

Tabelle 20: Realisierbarkeit der nicht funktionalen Anforderungen bei Team City

Nicht funktionale Anforderungen	
L00021-24-1	Team City sieht vor, dass Anwender Einstellungen über ein Web-Interface vornehmen [79].
L00021-24-2	Team City sieht vor, dass Anwender Ergebnisse über ein Web-Interface abrufen [80].
L00021-24-3	Team City lässt sich auf einem Windows Server 2008 System installieren [81][82].
L00021-24-4	Team City lässt sich auf einem Windows Server 2008 System installieren [83][84].
L00021-24-5	Team City kann mit zwei unterschiedlichen Lizenzen betrieben werden: Der kostenfreien *Professional Server Licence* und der kostenpflichtigen *Enterprise Server Licence* [85]:

Professional Server Licence (kostenlos)	Enterprise Server Licence (€ 1799)
- 20 *Build Configurations* - 3 *Build Agents*	- Unbegrenzte Anzahl an *Build Configurations* - 3 bis 100 *Build Agents*

6 Quantitative Bewertung

6.1 Vorbetrachtungen

Nachdem identifiziert wurde, welche Anforderungen von welchen CI-Servern erfüllt werden können, soll hier über eine qualitative Betrachtung hinaus eine quantitativ nachvollziehbare Bewertung erfolgen. Diese soll die Beurteilung darüber erleichtern, für welche Tools eine Testinstallation lohnenswert wäre. Es gilt also im Folgenden ein Bewertungssystem zu finden, bei dem die für die SMT-Entwicklungsumgebung am besten geeignete Lösung hoch priorisiert wird. Hierfür eignet sich ein Punktbewertungssystem, das Kriterien – in diesem Fall den Anforderungen aus dem Lastenheft – Erfüllungsgrade zuordnet. Es wird also eine Form der sogenannten *Punktwertmethode* für diese Auswertung eingesetzt.
Der Erfüllungsgrad zu einem Kriterium macht eine Aussage darüber, wie sich diese Anforderung umsetzen lässt, das heißt, mit welchem Aufwand bei der Umsetzung zu rechnen sein wird.
Zusätzlich lassen sich die Kriterien gewichten. Die Erfüllung einer hoch gewichteten Anforderung soll dementsprechend höher bewertet werden als die Erfüllung einer gering gewichteten. Hoch gewichtete Anforderungen sind beispielsweise solche, die im Lastenheft als unabdingbar (die Anforderung muss umgesetzt werden) identifizierbar sind. Gering gewichtete Anforderungen sind solche, die im Lastenheft als möglich (die Anforderung kann umgesetzt werden) identifizierbar sind.
Die Punkte, die sich aus einem Kriterium ergeben, setzen sich also zusammen aus der Relevanz der Anforderung sowie dessen Erfüllungsgrad. Das heißt konkret, dass für jede Anforderung festgelegt wird, wie wichtig diese ist (Relevanz) und wie anspruchsvoll deren Umsetzung ist (Erfüllungsgrad). Entsprechend wird eine erreichbare Punktzahl für die Relevanz (je wichtiger, desto höher) und den Erfüllungsgrad (je einfacher, desto höher) einer jeden Anforderung definiert:

Die **Relevanz** einer Anforderung ergibt sich aus deren Stellenwert in einem auf Kontinuierliche Integration umzustellenden Entwicklungsprozess und wird in drei verschiedene Grade respektive Punktzahlen unterteilt. Eine unabdingbare Anforderung hat die Relevanz 3, eine immer noch wichtige Anforderung hat die Relevanz 2 und eine mögliche Anforderung hat die Relevanz 1. Die Zuordnung der Punktzahlen erfolgt weiter unten.

Der **Erfüllungsgrad** soll die Umsetzbarkeit in fünf Stufen abbilden. Die Zuordnung von Punktzahlen zu verschiedenen Erfüllungsgraden erfolgt weiter unten.

- Durch Recherche nicht herauszufinden, Testinstallation für Bewertung der Umsetzbarkeit erforderlich
- Nicht umsetzbar
- Schwierig und nur mit Aufwand umsetzbar
- Relativ einfach umsetzbar, zum Beispiel mit Hilfe eines Plug-Ins
- Einfach umsetzbar, zum Beispiel *out of the box*

Eine Ausnahme bildet hierbei die nicht funktionale Anforderung 24-5, welche fordert, dass die Werkzeuge nach einem immerwährenden Lizenzmodell betrieben werden sollen und nach der einmalig Lizenzgebühren anfallen dürfen. In diesem Fall sollen kostenlose Lösungen in die Kategorie *einfach umsetzbar* und lizenzbasierte Lösungen in die Kategorie *relativ einfach umsetzbar* fallen. Eine stärkere Gewichtung zu Gunsten niedriger Kosten lässt das Lastenheft nicht zu, da ein kostenloses Modell genauso akzeptabel ist wie ein kostenpflichtiges. Eventuell muss bei ähnlichem Abschneiden von kostenlosen und kostenpflichtigen Lösungen dieser Punkt noch einmal gesondert und außerhalb der quantifizierten Bewertung betrachtet werden.

Es ergibt sich folgende Matrix, die alle möglichen Kombinationen aus Erfüllungsgrad und Relevanz einer Anforderung abdeckt. Aus dieser geht hervor, mit welcher Punktzahl eine Kombinationen zu bewerten ist:

Tabelle 21: Bewertungssystem bei Normalgewichtung (siehe unten) von Relevanz und Anforderungen (Gewichtung 1)

	Muss = 3	Sollte = 2	Kann = 1
Einfach (*out of the box*) = 3	9	6	3
Noch einfach (z.B. Plug-In) = 2	6	4	2
Nur mit Aufwand = 1	3	2	1
Durch Recherche nicht absehbar = 1	3	2	1
Nicht umsetzbar = 0	0	0	0

Wie in Tabelle 21 ersichtlich, ergibt sich die Punktzahl für jede Anforderung aus der Multiplikation der Punktzahlen von Relevanz und Erfüllungsgrad. Diese Art der Bewertung trägt der Tatsache Rechnung, dass die Punktzahl für die Umsetzung einer Anforderung eine *Kombination* aus der Relevanz und dem Erfüllungsgrad darstellt.
Zusätzlich ergibt sich hierdurch die Möglichkeit, verschiedene *Gewichtungen* zu definieren, mit deren Hilfe Aussagen zu verschiedenen Aspekten eines *Build-Servers* gemacht werden können.

Bei der oben gezeigten *Normalgewichtung* geht die Erfüllung einer Anforderung etwa gleich stark in die Bewertung mit ein wie deren Relevanz. So bekommt eine unwichtige Anforderung mit der Relevanz 1, die einfach umsetzbar ist, ebenso viele Punkte wie eine nur mit Aufwand umsetzbare unabdingbare Anforderung (*kann* und *einfach* entspricht derselben Punktzahl wie *muss* und *nur mit Aufwand*).
Denkbar wären weiterhin andere Gewichtungen. Diese würden eine genauere Aussage über entweder die Erfüllung oder die Relevanz einer Anforderung erlauben. Neben der oben vorgestellten *Normalgewichtung* sollen noch weitere Gewichtungen ausgewertet werden. So muss zum Beispiel beachtet werden, dass im ungünstigsten Fall eine Anforderung, deren Umsetzbarkeit sich durch Recherche nicht abschätzen lässt, *gar nicht* umsetzbar sein könnte. Bei dieser Gewichtung soll also der Punkt *durch Recherche nicht absehbar* mit null Punkten bewertet werden. Das Ergebnis dieser Gewichtung zeigt also, bei welchen *Build-Servern* Unsicherheiten bestehen.

Tabelle 22: Bewertungssystem bei Normalgewichtung von Relevanz und Anforderungen mit Worst-Case-Annahme bei durch Recherchen nicht absehbaren Anforderungen (Gewichtung 2)

	Muss = 3	Sollte = 2	Kann = 1
Einfach (*out of the box*) = 3	9	6	3
Noch einfach (z.B. Plug-In) = 2	6	4	2
Nur mit Aufwand = 1	3	2	1
Durch Recherche nicht absehbar = 0	0	0	0
Nicht umsetzbar = 0	0	0	0

Durch die letzte Gewichtungsart soll feststellbar werden, welcher *Build-Server* die *meisten* Anforderungen umsetzen kann. Hierbei interessiert nicht der Erfüllungsgrad, sondern lediglich *ob* eine Anforderung überhaupt umsetzbar ist. Dazu werden *alle* Erfüllungsgrade, bei denen eine Anforderung umsetzbar ist, jeweils mit einem Punkt bewertet. Hier interessiert also nicht, ob eine Anforderung einfach oder aufwendig umzusetzen ist; die meisten Punkte erhält der *Build-Server*, mit dem schlicht die *meisten* Anforderungen umsetzbar sind.

Tabelle 23: Bewertungssystem bei Anforderungsgewichtung (Gewichtung 3)

	Muss = 3	Sollte = 2	Kann = 1
Einfach (*out of the box*) = 1	3	2	1
Noch einfach (z.B. Plug-In) = 1	3	2	1
Nur mit Aufwand = 1	3	2	1
Durch Recherche nicht absehbar = 0	0	0	0
Nicht umsetzbar = 0	0	0	0

6.2 Bewertung

Auf Grundlage dieser Bewertungssysteme sowie den im vorangegangenen Kapitel bestimmten Erfüllungsgraden kann nun berechnet werden, wie gut der jeweilige *Build-Server* die Anforderungen umsetzt. In Anhang A finden sich die zugehörigen Excel-Tabellen, mit deren Hilfe für jede der drei oben vorgestellten Gewichtungsarten ein Wert ermittelt wurde und eine prozentuale Aussage trifft. Ein Wert von 100 % würde dementsprechend bedeuten, dass dieser *Build-Server alle* Anforderungen mit dem Erfüllungsgrad *Einfach* erfüllt.

Die folgende Tabelle 24 zeigt das Abschneiden der verschiedenen *Build-Server* in Abhängigkeit der Gewichtung. Zu jeder Gewichtung findet sich die Tabelle in Anhang B, welche die Berechnungsgrundlage darstellt. Als abschließende Bewertung wird ein Mittelwert über die drei Gewichtungsarten gebildet. Dadurch, dass eine *Anforderungsgewichtung* (Gewichtung 3), jedoch keine Gewichtung des Erfüllungsgrades mit eingeht, verschiebt diese Durchschnittsbildung die Gesamtwertung in Richtung Anforderungsgewichtung. Dies ist so beabsichtigt, da diese Art der Gewichtung als relevanter erachtet wird.

Tabelle 24: Bewertung der *Build-Server* – für Berechnungen siehe Anhang A Tabellen 1 – 6

	Hudson	Jenkins	Bamboo	TFS	Cruise Control	Team City
Gewichtung 1	82,46 %	82,46 %	81,29 %	66,67 %	72,51 %	75,44 %
Gewichtung 2	76,02 %	76,02 %	75,44 %	58,48 %	64,91 %	67,84 %
Gewichtung 3	80,70 %	80,70 %	82,46 %	70,18 %	68,42 %	75,44 %
Mittelwert	**79,73 %**	**79,73 %**	**79,73 %**	**65,11 %**	**68,61 %**	**72,91 %**

Diese Mittelwerte sollen als quantifizierte Rechercheergebnisse betrachtet werden – auf dieser Grundlage soll nun entschieden werden, für welche *Build-Server* eine Testinstallation durchzuführen ist.

Die errechneten Werte decken sich mit den Erwartungen aus der Recherche. Hudson und Jenkins schneiden wegen ihrer großen Ähnlichkeit identisch ab. Beide erzielen eine hohe Wertung und sollen daher näher untersucht werden. Dies kommt einer eventuellen endgültigen Auswahl insofern entgegen, als aufgrund der Faktenlage keines der beiden Tools definitiv vorzuziehen ist.
Bamboo schneidet mit exakt derselben Bewertung ab wie Hudson und Jenkins – lediglich bei den unterschiedlichen Gewichtungen lassen sich geringe Unterschiede ausmachen. Vergleicht man also über den Mittelwert hinaus auch die einzelnen Gewichtungen bei Bamboo, Hudson und Jenkins, so kommt man zu dem Schluss, dass Bamboo durch die höhere Punktzahl bei Gewichtung 3 insgesamt mehr

Anforderungen erfüllen kann. Die Umsetzung ist für diesen Fall allerdings minimal aufwendiger. Bamboo soll ebenfalls mittels einer Testinstallation weitergehend bewertet werden.
Auch Team City schneidet gut ab, allerdings mit auffallend niedriger Wertung bei Gewichtung 2. Es gibt somit einige Unsicherheiten hinsichtlich der Umsetzbarkeit gewisser Anforderungen. Um diese auszuräumen und wegen der relativ hohen Bewertung soll auch Team City mittels einer Testinstallation weitergehend bewertet werden.
Wie sich bereits in der qualitativen Bewertung abgezeichnet hat, erhalten der Team Foundation Server und Cruise Control die niedrigsten Bewertungen. Der Team Foundation Server ist als reines CI-Tool zu sperrig, da er weit mehr als nur die Aufgaben eines reinen CI-Tools übernehmen kann. Der Fokus auf eine Microsoft-Entwicklungsumgebung stellt ein weiteres Hindernis dar. Mit nur 58,48 % in Gewichtung 2 gibt es außerdem zu viele Unsicherheiten was die Umsetzbarkeit der Anforderungen betrifft. Von einer weitergehenden Bewertung von Team Foundation Server und Cruise Control wird aus diesen Gründen abgesehen.

7 Testinstallation

7.1 Vorbetrachtungen

In den beiden vorangegangenen Kapiteln wurden zuerst qualitativ und anschließend quantitativ die ausgewählten *Build-Server* bewertet. Dabei hat sich die in Betracht zu ziehende Auswahl auf vier reduziert: *Hudson*, *Jenkins*, *Bamboo* und *Team City*. Diese sollen nun im Folgenden auf einer Testumgebung (siehe hierzu Abschnitt 7.2) installiert und getestet werden. Hierbei soll in erster Linie überprüft werden, ob sich die Ergebnisse dieser praktischen Auswertung mit denen aus der qualitativen Beurteilung decken. Es wird also für jede der Testinstallationen geprüft, ob die vier Anwendungsfälle sowie die funktionalen- und nicht funktionalen Anforderungen umsetzbar sind. Am Ende dieses Kapitels soll sich ein vollständiges Bild aller überprüften *Build-Server* ergeben. Es soll Kapitel 5 dort ergänzen, wo noch Unsicherheiten bestehen und darüber hinaus zeigen, inwieweit sich die recherchierten Ergebnisse mit den nun praktisch zu ermittelnden decken. Da nach Abschluss dieser Testinstallation alle Anforderungen überprüft sein werden, ergibt sich die Notwendigkeit einer erneuten quantitativen Beurteilung, weil der Erfüllungsgrad *durch Recherche nicht herauszufinden* entfällt.
Zusätzlich kann bei der praktischen Beurteilung der Anforderungen auch ein erster Überblick über die Usability gewonnen werden. Diese spielt sowohl bei der Einrichtung als auch im täglichen Gebrauch eine Rolle.
Am Ende dieses Kapitels soll also eine belastbare Entscheidungsgrundlage für einen *Build-Server* stehen.

7.2 Testumgebung

Hierbei handelt es sich um einen Desktop-PC, auf dem Windows 7 Professional 64 Bit mit Service Pack 1 läuft. Da ein Server für Testzwecke nicht zur Verfügung steht, wird ausschließlich auf dem *Localhost* getestet. Die Anforderungen L00021-24-3 und L00021-24-4, die fordern, dass die zu installierenden Werkzeuge auf einem zu Windows Server 2008 kompatiblen bzw. auf einem virtuellen Server zu installieren seien, lassen sich somit nicht überprüfen.

Computer: SMTDT143
CPU: Intel® Core™ i5-2400
RAM: 8,00 GB

7.3 Hudson

Hudson wird auf dem Localhost installiert und getestet. Hierfür wird ein *WAR-File* von der Hudson-Webseite herunter geladen, das neben der aktuellsten Hudson-Version auch einen *Servlet-Container* beinhaltet, in den die Webapplikation Hudson ausgebracht werden kann. Die hier getestete Version hat die Versionsnummer 3.2.0. Nach dem Herunterladen kann Hudson also direkt aus der Konsole heraus gestartet werden:

```
cd Downloads
java –jar Hudson-3.2.0.war
```

Hudson ist standardmäßig an Port 8080 zu erreichen – dieser kann nun natürlich geändert werden. Zudem lässt sich Hudson als Windows-Dienst einrichten, um den oben beschriebenen Startprozess zu vermeiden: Hudson wird so auf Wunsch direkt nach dem Start des Betriebssystems geladen.

Der Zugriff sowie die Konfiguration erfolgt über einen Webbrowser:

```
localhost:8080
```

Hier gelangt man auf die Hudson-Startseite, von der aus man eine Übersicht über alle Projekte erhält und in die Projekt- sowie die allgemeine Verwaltung wechseln kann (siehe Abb. 1).

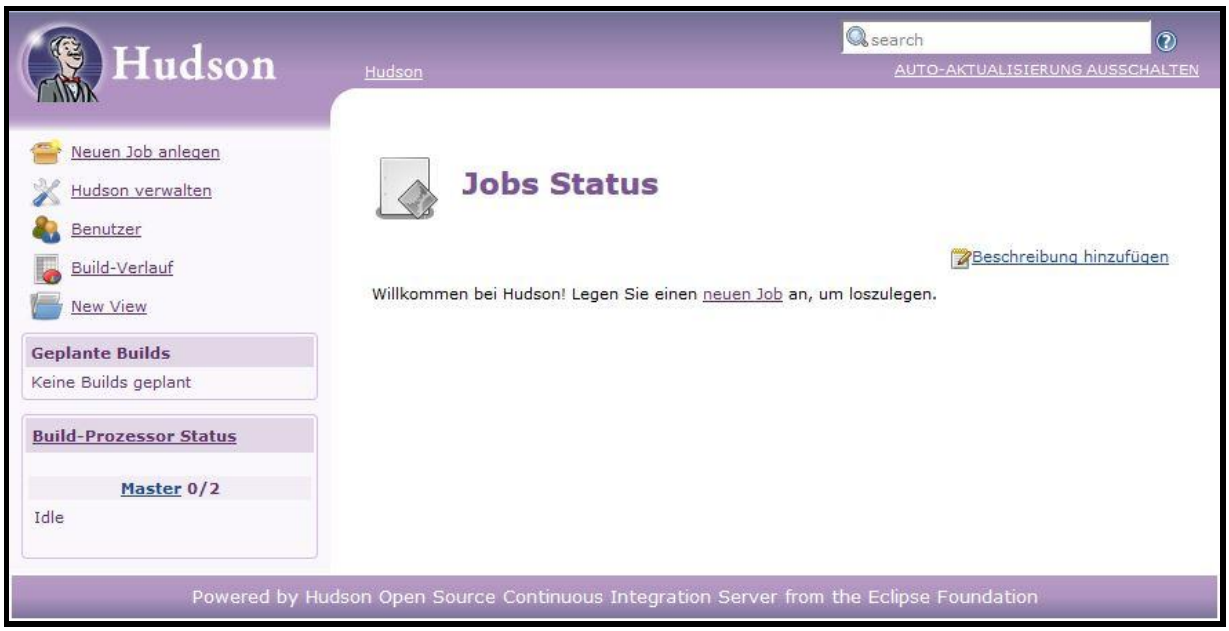

Abbildung 1: Hudson-Startseite

Zur Überprüfung der Funktionalität von Hudson soll im ersten Schritt der Buildvorgang mit statischer Codeanalyse durchgeführt werden. Die Erfüllung der weiteren Anforderungen soll während der Erstellung und Auswertung dieses neuen Jobs ebenfalls überprüft werden.

Vor dem Anlegen des neuen Jobs wurden folgende Plug-Ins installiert:

- Hudson JIRA Plug-In
- Hudson Subversion Plug-In
- Polarion Plug-In
- Warnings Plug-In

Die verfügbaren Plug-Ins lassen sich sehr einfach installieren und stehen nach einem Neustart von Hudson zur Verfügung. Der neu angelegte Job kann nun konfiguriert werden. Die folgenden Abbildungen zeigen die Konfiguration eines Beispielsjobs. Die dazugehörigen Tabellen verknüpfen Anwendungsfälle oder Anforderungen mit diesen Abbildungen.

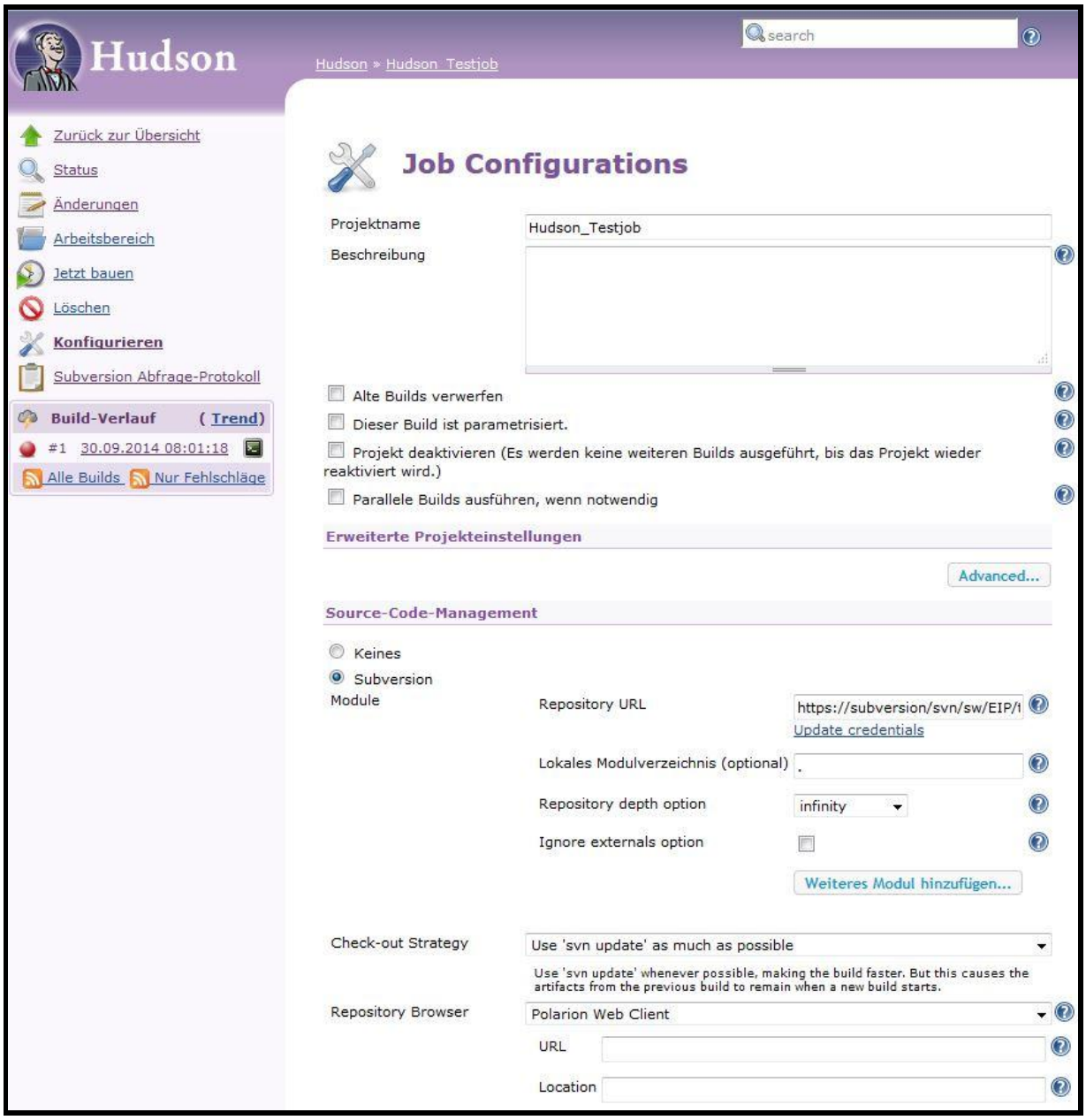

Abbildung 2: Hudson Job Konfiguration 1/4

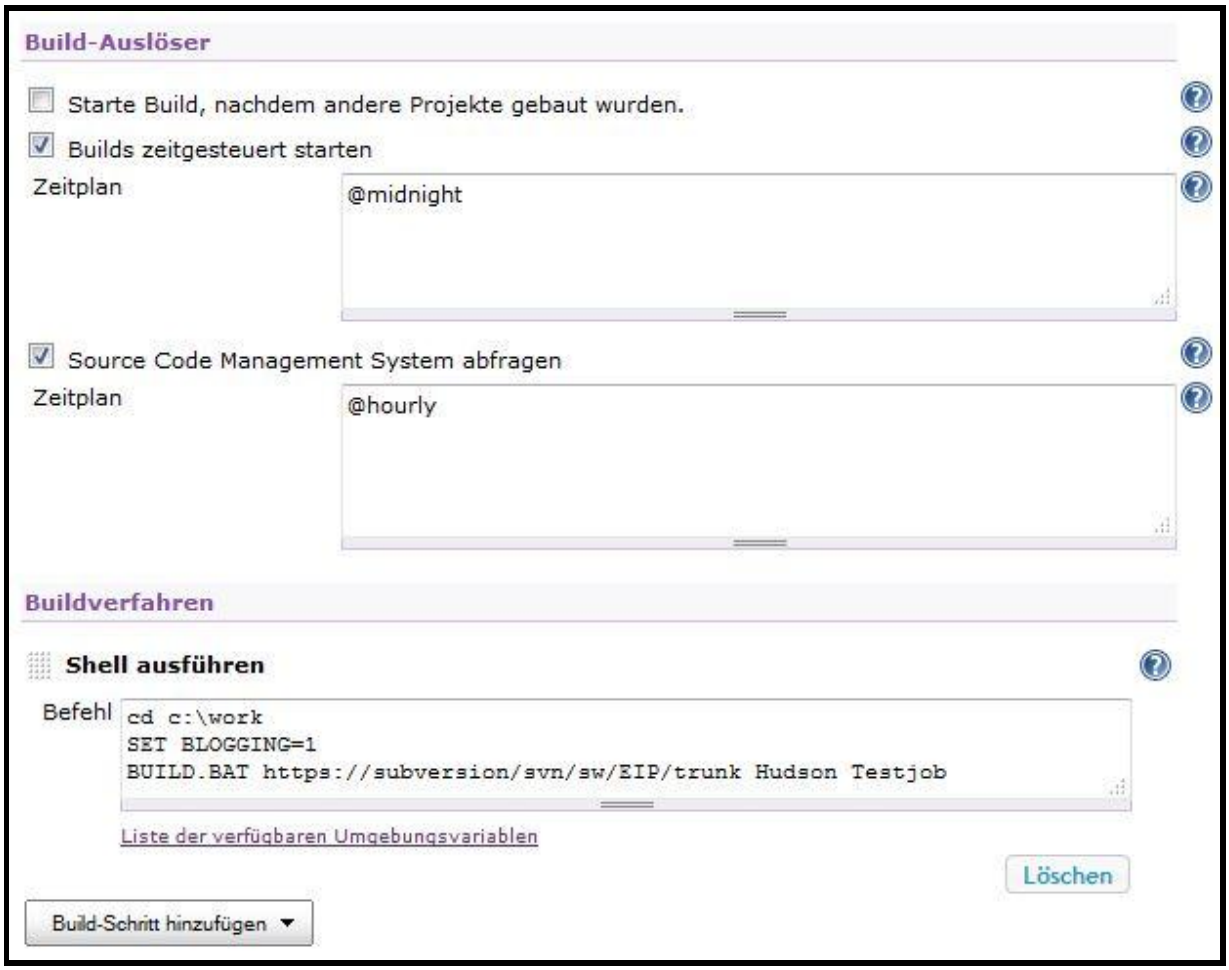

Abbildung 3: Hudson Job Konfiguration 2/4

Tabelle 25: In Abbildung 3 umgesetzte Anwendungsfälle und Anforderungen bei Hudson

Zu Abbildung 2 und 3	
Einstellung	**Anwendungsfall / Anforderung**
Build Auslöser – Builds zeitgesteuert starten	L00021-23-3
Source-Code-Management – Repository Browser	L00021-23-5
Build Auslöser – Source Code Management System abfragen	L00021-23-6
Buildverfahren – Shell ausführen	Anwendungsfall 1

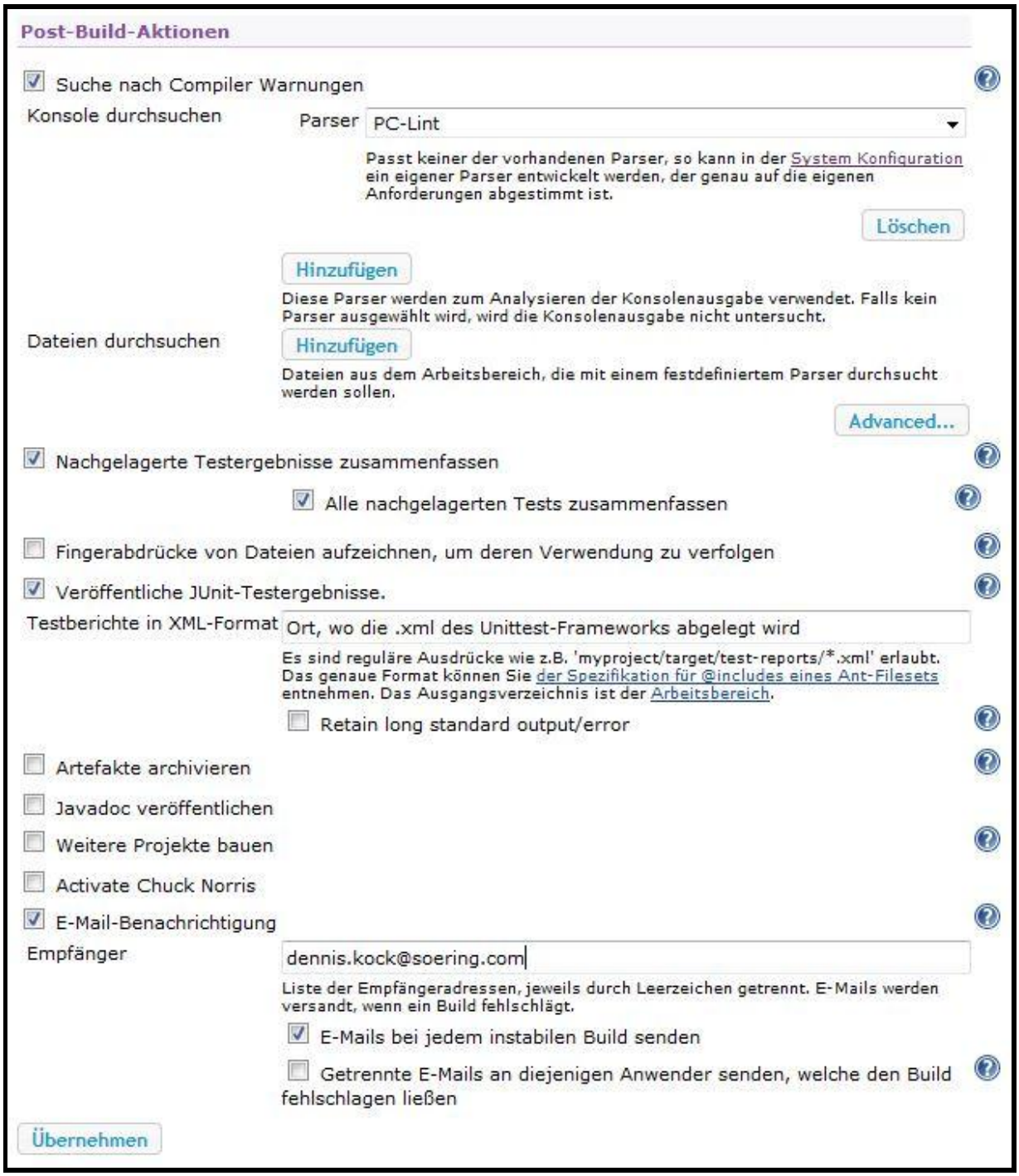

Abbildung 4: Hudson Job Konfiguration 3/4

Tabelle 26: In Abbildung 4 umgesetzte Anwendungsfälle und Anforderungen bei Hudson

Zu Abbildung 4	
Einstellung	**Anwendungsfall / Anforderung**
Post-Build-Aktionen – Suche nach Compiler Warnungen	Anwendungsfall 2
Post-Build-Aktionen – Veröffentliche JUnit-Testergebnisse	Anwendungsfall 3

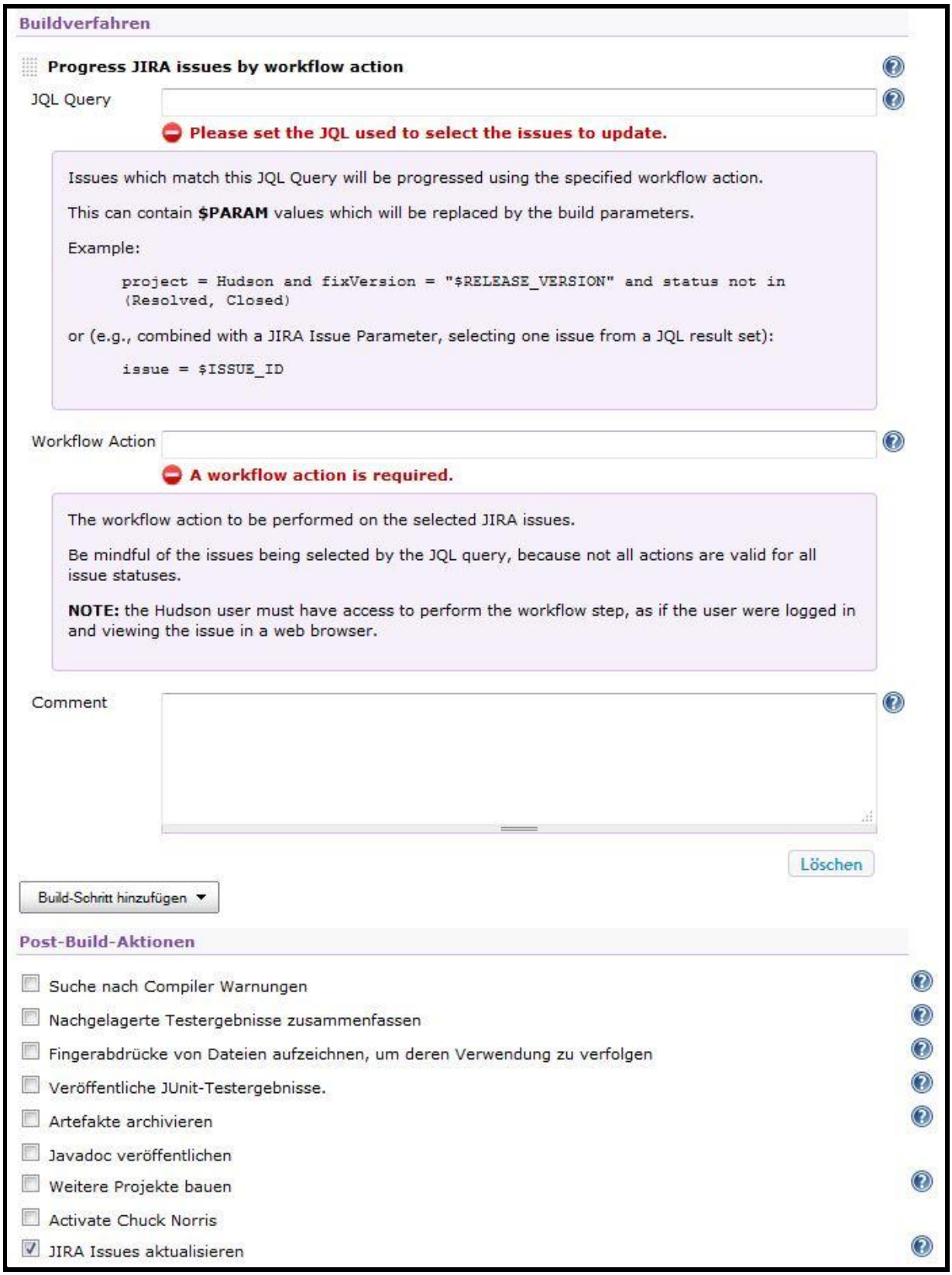

Abbildung 5: Hudson Job Konfiguration 4/4

Tabelle 27: In Abbildung 5 umgesetzte Anwendungsfälle und Anforderungen bei Hudson

Zu Abbildung 5	
Einstellung	**Anwendungsfall / Anforderung**
Buildverfahren – Progress JIRA issues by workflow action Post-Build-Aktionen – JIRA-Issues aktualisieren	L00021-23-13

Tabelle 28: Umsetzung der übrigen Anwendungsfälle und Anforderungen bei Hudson

Anwendungsfall / Anforderung	**Erfüllt / nicht erfüllt**	**Begründung**
Anwendungsfall 4	Ausstehend	Die Interaktion mit den Schnittstellen des zu testenden Systems geschieht skriptbasiert. Hudson unterstützt die Ausführung von Skripten, mit Hilfe derer die zu testenden Systeme angesprochen werden könnten. Etwaige Testergebnisse müssten von Hudson entgegen genommen werden – beispielsweise über die Konsole oder über eine XML-Datei, die zu erzeugen wäre.
L00021-23-1	Erfüllt	Die Zuordnung von Anwendungsfällen zu Jobs wurde überprüft.
L00021-23-2	Erfüllt	Es gibt keine Limitierung der neu anlegbaren Jobs.
L00021-23-4	Erfüllt	Beispiele für ereignisgesteuerte Auslöser sind ein Commit in das Repository oder die Option *Starte Build, nachdem andere Projekte gebaut wurden.*
L00021-23-7	Nicht erfüllt	Das Plug-In *Hudson/Jenkins Mylyn Builds Connector* für Eclipse lässt sich zwar installieren, jedoch funktioniert es mit der eingesetzten Eclipse-Version[5] nicht wie beschrieben.
L00021-23-8	Erfüllt	Hudson bietet Möglichkeiten, aktuelle Job-Ergebnisse zu präsentieren.
L00021-23-9	Erfüllt	Hudson bietet Möglichkeiten, in der Vergangenheit erzeugte Job-Ergebnisse zu präsentieren.
L00021-23-10	Nicht erfüllt	Es ist weder in Hudson direkt noch durch ein Plug-In vorgesehen, Job-Ergebnisse in einem zu Microsoft Office 2010 kompatiblen Format zu dokumentieren.
L00021-23-11	Nicht erfüllt	Es ist weder in Hudson direkt noch durch ein Plug-In vorgesehen, Job-Ergebnisse im PDF-Format zu dokumentieren.
L00021-23-12	Ausstehend	Diese Anforderung muss bewertet werden, sobald Polarion in der SMT-Softwareentwicklung eingesetzt wird.
L00021-24-1	Erfüllt	Die Hudson-Oberfläche wurde sowohl in Firefox als auch im Internet Explorer 10 aufgerufen und geprüft.
L0021-24-2	Erfüllt	Die Hudson-Oberfläche wurde sowohl in Firefox als auch im Internet Explorer 10 aufgerufen und geprüft.
L0021-24-3	Nicht anwendbar	Diese Anforderung lässt sich nicht überprüfen, da kein Testserver zur Verfügung steht.
L0021-24-4	Nicht anwendbar	Diese Anforderung lässt sich nicht überprüfen, da kein Testserver zur Verfügung steht.
L0021-24-5	Erfüllt	Die Benutzung von Hudson ist kostenlos möglich.

[5] Getestet auf Eclipse mit der Version Indigo Service Release 2 (Build id: 20120216-1857)

7.4 Jenkins

Jenkins wird auf dem Localhost installiert und getestet. Neben einem War-File steht auch ein Installationspaket mit einer *setup.exe* zur Verfügung. Nach deren Ausführung steht Jenkins als Windows-Dienst bereit.

Abbildung 6: Jenkins-Startseite

Zur Überprüfung der Funktionalität von Jenkins soll im ersten Schritt der Buildvorgang mit statischer Codeanalyse durchgeführt werden. Die Erfüllung der weiteren Anforderungen soll während der Erstellung und Auswertung dieses neuen Jobs ebenfalls überprüft werden.

Vor dem Anlegen des neuen Jobs wurden folgende Plug-Ins installiert:

- JIRA Plug-In
- Polarion Plug-In
- Subversion Plug-In
- Warnings Plug-In

Die verfügbaren Plug-Ins lassen sich sehr einfach installieren, der Neustart von Jenkins kann sogar direkt aus der Weboberfläche erfolgen und muss nicht manuell durchgeführt werden. Diese Tatsache sowie das Vorhandensein der *setup.exe* für die Installation machen das Aufsetzen von Jenkins noch einfacher als bei Hudson.

Der neu angelegte Job kann nun konfiguriert werden. Die folgenden Abbildungen zeigen die Konfiguration eines Beispielsjobs. Die dazugehörigen Tabellen verknüpfen Anwendungsfälle oder Anforderungen mit dieser Abbildung.

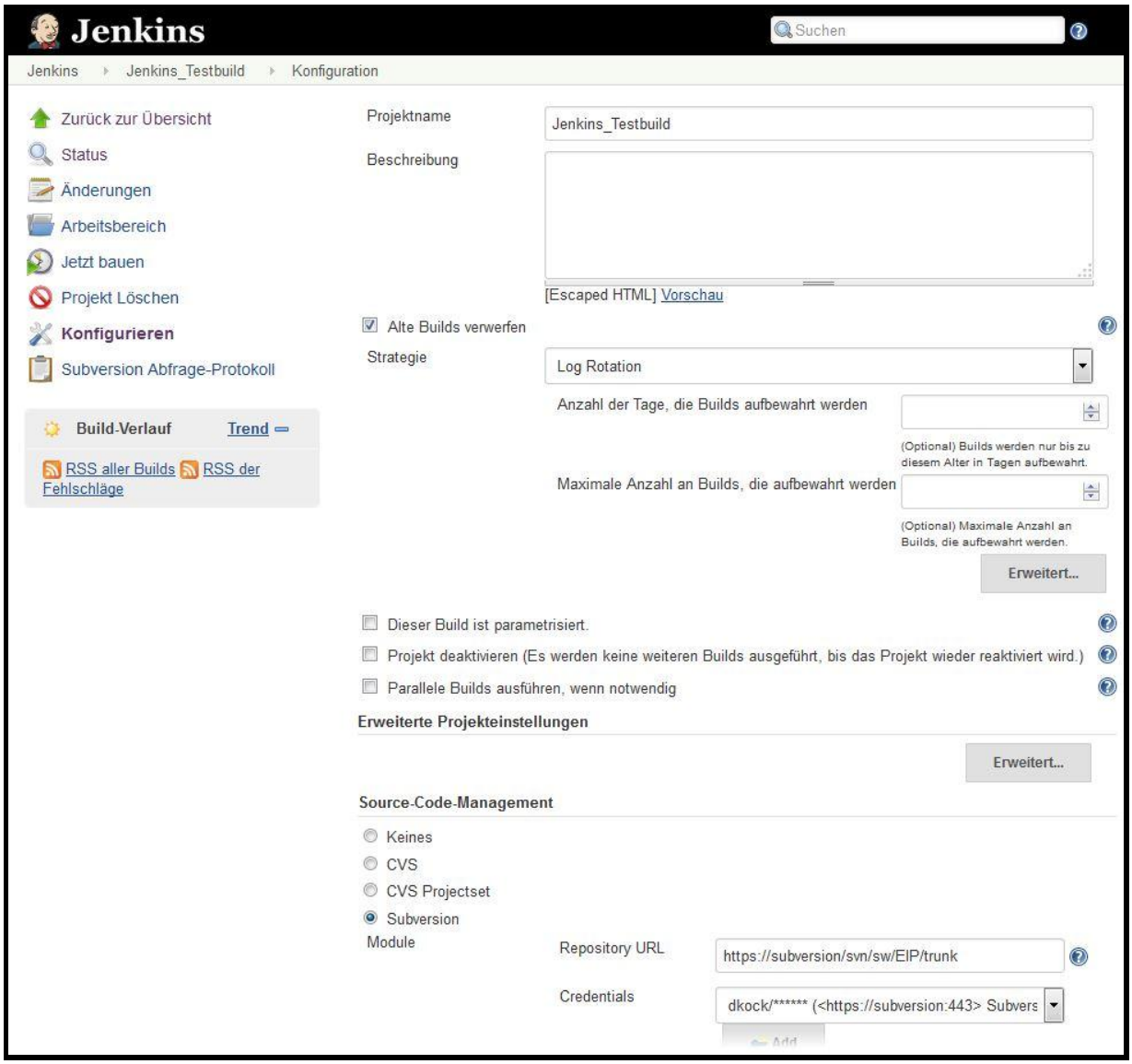

Abbildung 7: Jenkins Job Konfiguration 1/4

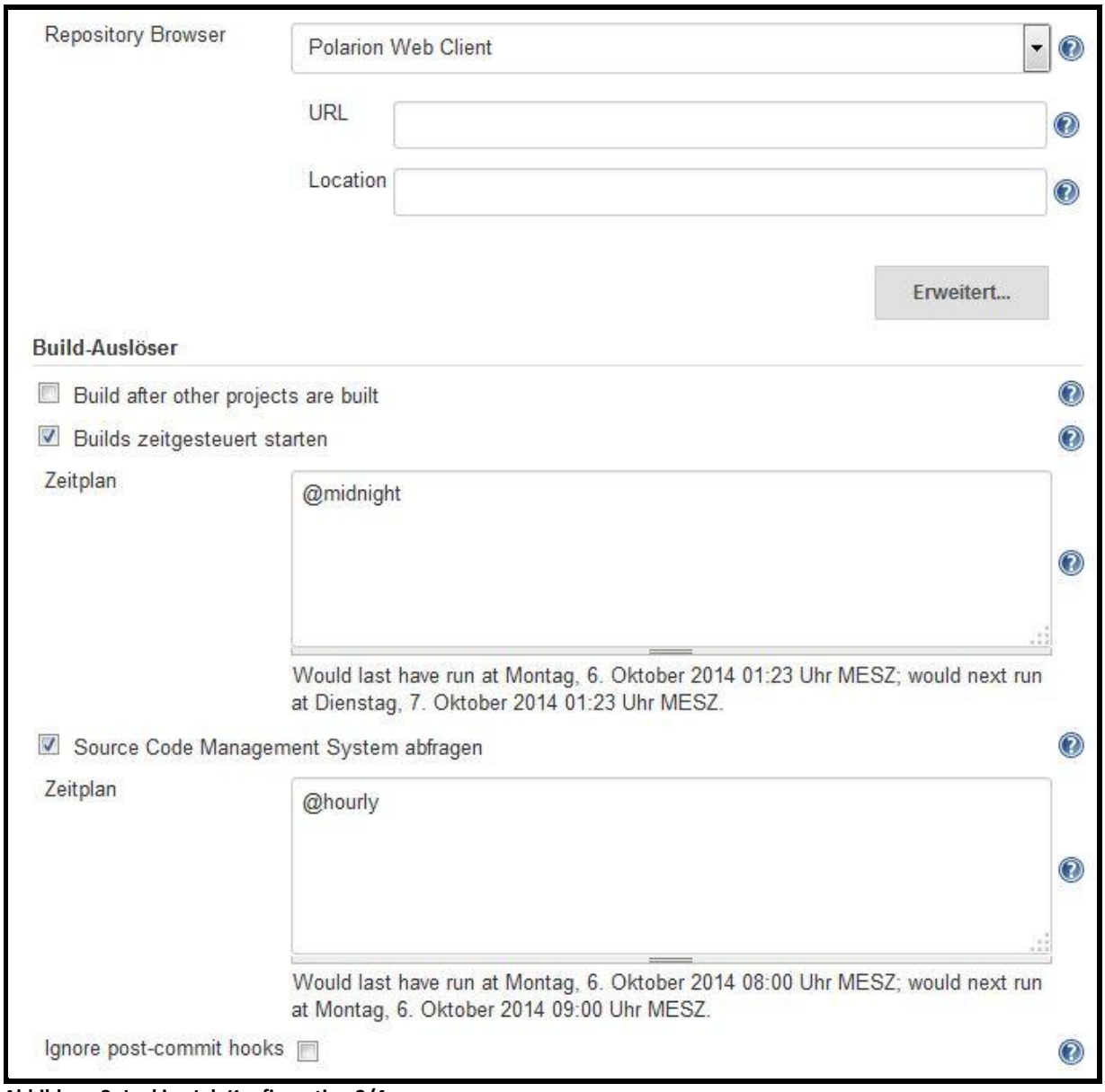

Abbildung 8: Jenkins Job Konfiguration 2/4

Tabelle 29: In Abbildung 7 umgesetzte Anwendungsfälle und Anforderungen bei Jenkins

Zu Abbildung 8	
Einstellung	**Anwendungsfall / Anforderung**
Repository Browser	L00021-23-5
Builds zeitgesteuert starten	L00021-23-3
Source Code Management System abfragen	L00021-23-6

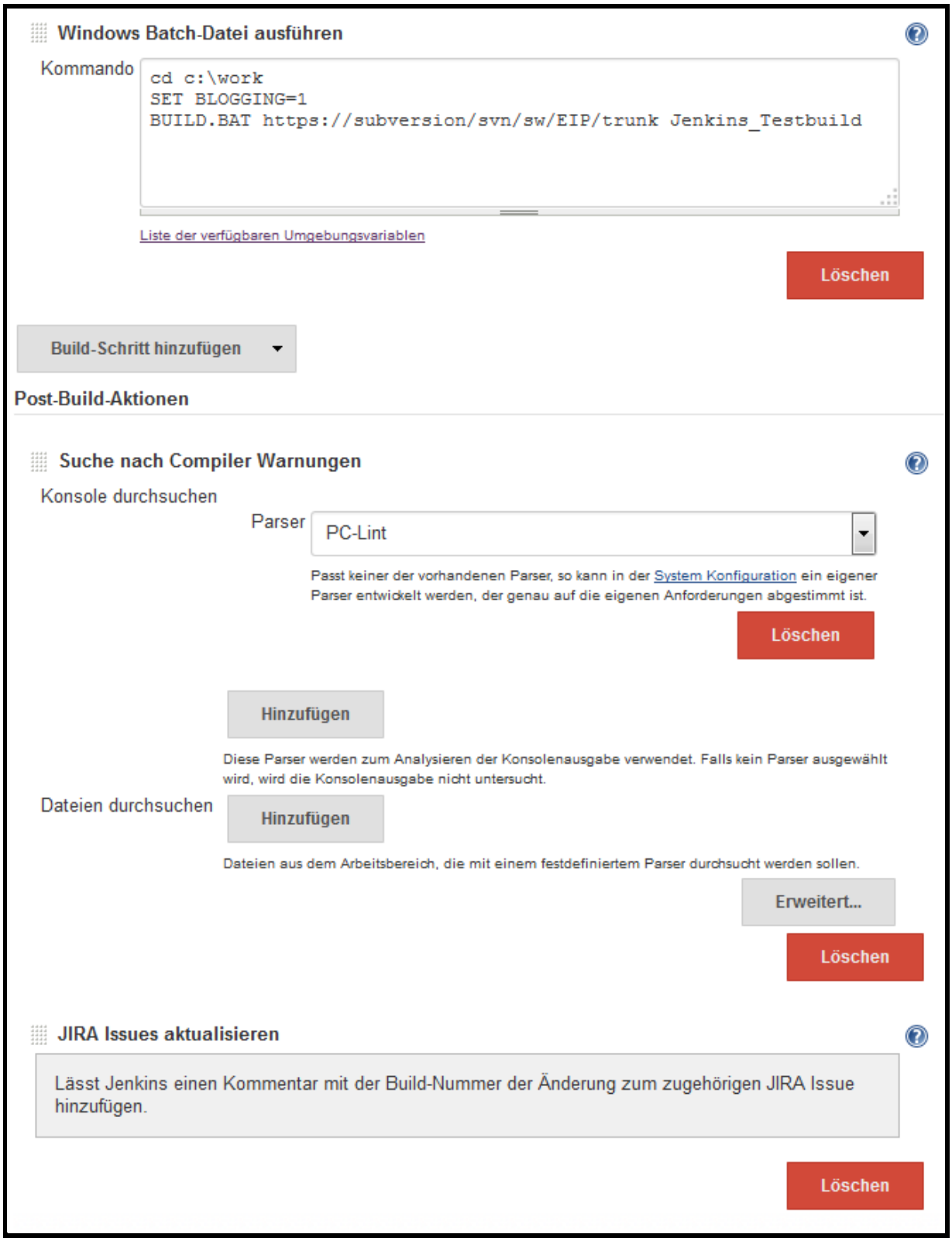

Abbildung 9: Jenkins Job Konfiguration 3/4

Tabelle 30: In Abbildung 9 umgesetzte Anwendungsfälle und Anforderungen bei Jenkins

Zu Abbildung 9	
Einstellung	**Anwendungsfall / Anforderung**
Windows Batch Datei ausführen	Anwendungsfall 1
Post-Build-Aktionen – Suche nach Compiler Warnungen	Anwendungsfall 2

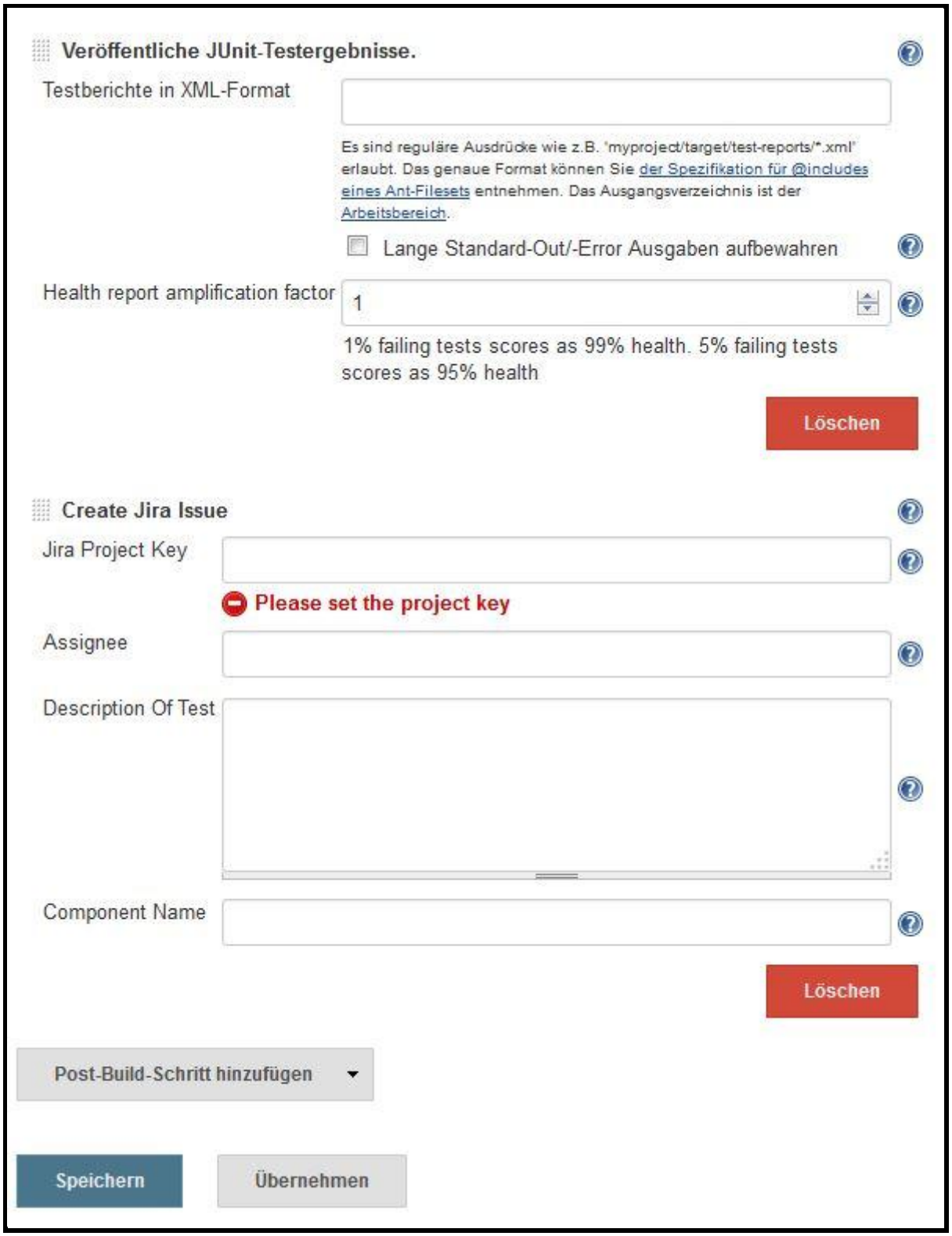

Abbildung 10: Jenkins Job Konfiguration 4/4

Tabelle 31: In Abbildung 10 umgesetzte Anwendungsfälle und Anforderungen bei Jenkins

Zu Abbildung 10	
Einstellung	**Anwendungsfall / Anforderung**
Veröffentliche JUnit-Testergebnisse	Anwendungsfall 3
Create Jira Issue	L00021-23-13

Tabelle 32: Umsetzung der übrigen Anwendungsfälle und Anforderungen bei Jenkins

Anwendungsfall / Anforderung	**Erfüllt / nicht erfüllt**	**Begründung**
Anwendungsfall 4	Ausstehend	Die Interaktion mit den Schnittstellen des zu testenden Systems geschieht skriptbasiert. Jenkins unterstützt die Ausführung von Skripten, mit Hilfe derer die zu testenden Systeme angesprochen werden könnten. Etwaige Testergebnisse müssten von Jenkins entgegen genommen werden – beispielsweise über die Konsole oder über eine XML-Datei, die zu erzeugen wäre.
L00021-23-1	Erfüllt	Die Zuordnung von Anwendungsfällen zu Jobs wurde überprüft.
L00021-23-2	Erfüllt	Es gibt keine Limitierung der neu anlegbaren Jobs.
L00021-23-4	Erfüllt	Beispiele für ereignisgesteuerte Auslöser sind ein Commit in das Repository oder die Option *Starte Build, nachdem andere Projekte gebaut wurden.*
L00021-23-7	Nicht erfüllt	Das Plug-In *Hudson/Jenkins Mylyn Builds Connector* für Eclipse lässt sich zwar installieren, jedoch funktioniert es mit der eingesetzten Eclipse-Version[6] nicht wie beschrieben.
L00021-23-8	Erfüllt	Jenkins bietet Möglichkeiten, aktuelle Job-Ergebnisse zu präsentieren.
L00021-23-9	Erfüllt	Hudson bietet Möglichkeiten, in der Vergangenheit erzeugte Job-Ergebnisse zu präsentieren.
L00021-23-10	Nicht erfüllt	Es ist weder in Jenkins direkt noch durch ein Plug-In vorgesehen, Job-Ergebnisse in einem zu Microsoft Office 2010 kompatiblen Format zu dokumentieren.
L00021-23-11	Nicht erfüllt	Es ist weder in Jenkins direkt noch durch ein Plug-In vorgesehen, Job-Ergebnisse im PDF-Format zu dokumentieren.
L00021-23-12	Ausstehend	Diese Anforderung muss bewertet werden, sobald Polarion in der SMT-Softwareentwicklung eingesetzt wird.
L00021-24-1	Erfüllt	Die Jenkins-Oberfläche wurde sowohl in Firefox als auch im Internet Explorer 10 aufgerufen und geprüft.
L0021-24-2	Erfüllt	Die Jenkins-Oberfläche wurde sowohl in Firefox als auch im Internet Explorer 10 aufgerufen und geprüft.
L0021-24-3	Nicht anwendbar	Diese Anforderung lässt sich nicht überprüfen, da kein Testserver zur Verfügung steht.
L0021-24-4	Nicht anwendbar	Diese Anforderung lässt sich nicht überprüfen, da kein Testserver zur Verfügung steht.
L0021-24-5	Erfüllt	Die Benutzung von Jenkins ist kostenlos möglich.

[6] Getestet auf Eclipse mit der Version Indigo Service Release 2 (Build id: 20120216-1857)

7.5 Bamboo

Die Installation von Bamboo gestaltet sich etwas aufwendiger als bei Hudson oder Jenkins. Es muss zusätzlich *Java JDK* installiert werden und eine *JAVA_HOME environment variable* gesetzt werden. Bamboo wird auf dem Localhost installiert.
Nachdem die Installation abgeschlossen ist, muss ein neuer *Plan* erzeugt werden, um mit Bamboo arbeiten zu können. Ein *Plan* in Bamboo definiert den gesamten Build-Prozess – jeder *Plan* hat einen *default job*, der diesem zugeordnet ist. Ein *Plan* besteht aus mehreren *Tasks*, die definiert werden können und dann nacheinander abgearbeitet werden. Ein *Task* ist beispielsweise das Auschecken des *Source Codes* aus dem *Repository*.

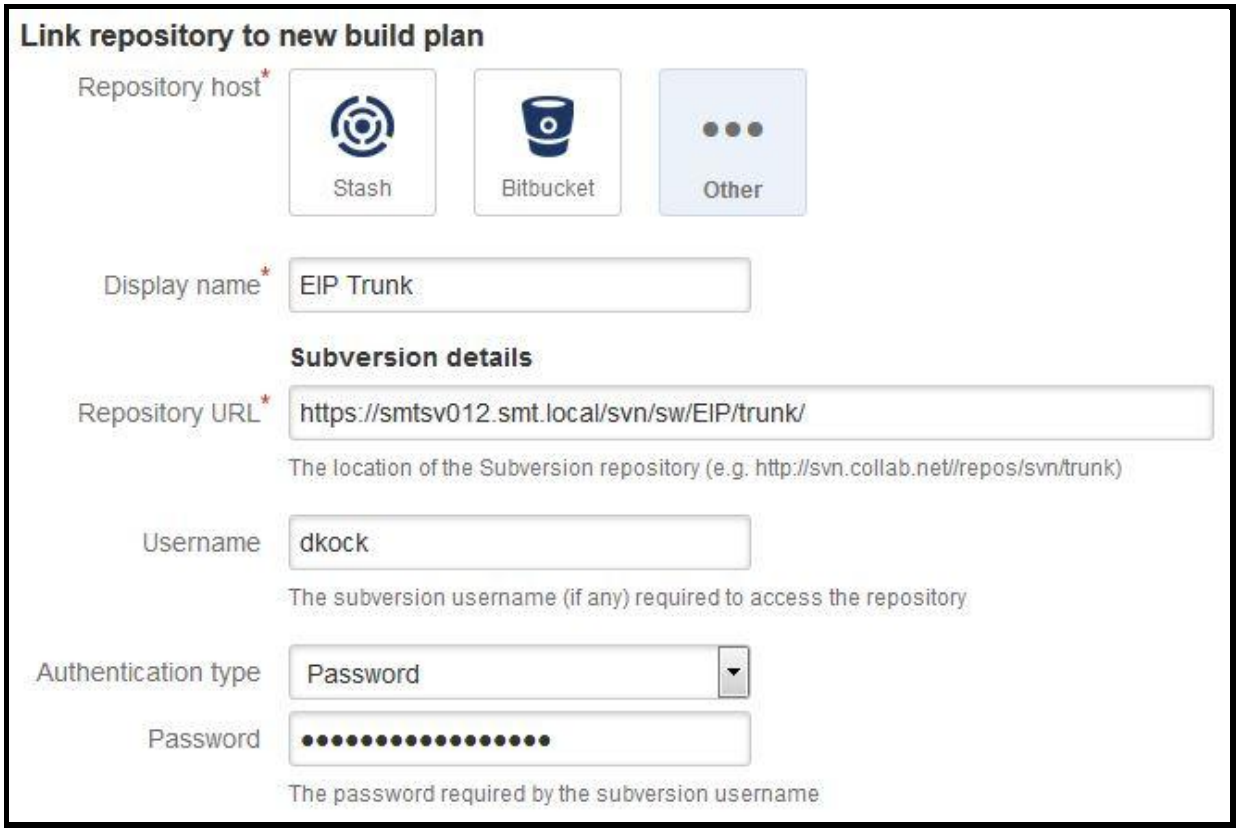

Abbildung 11: Bamboo Plankonfiguration 1/4

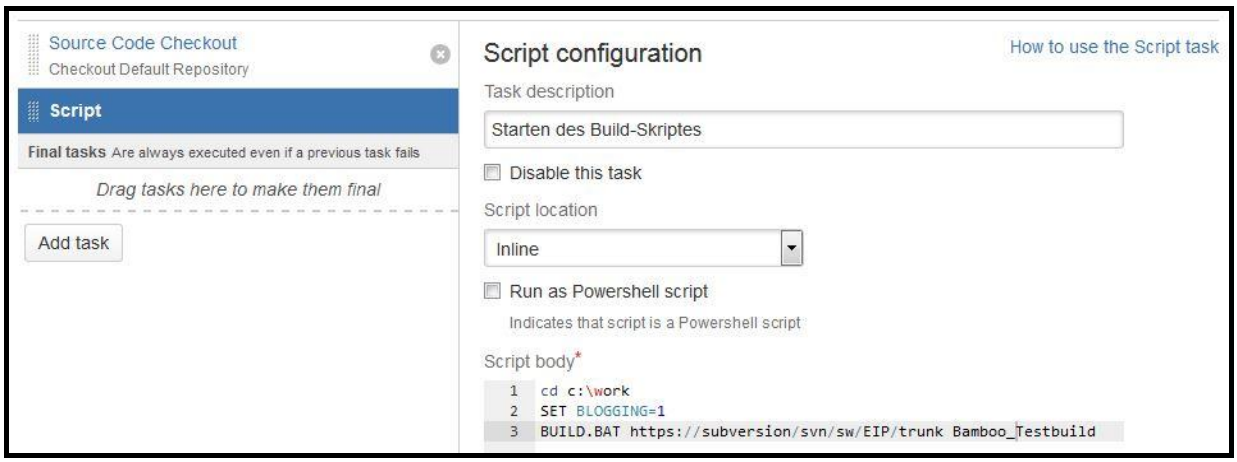

Abbildung 12: Bamboo Plankonfiguration 2/4

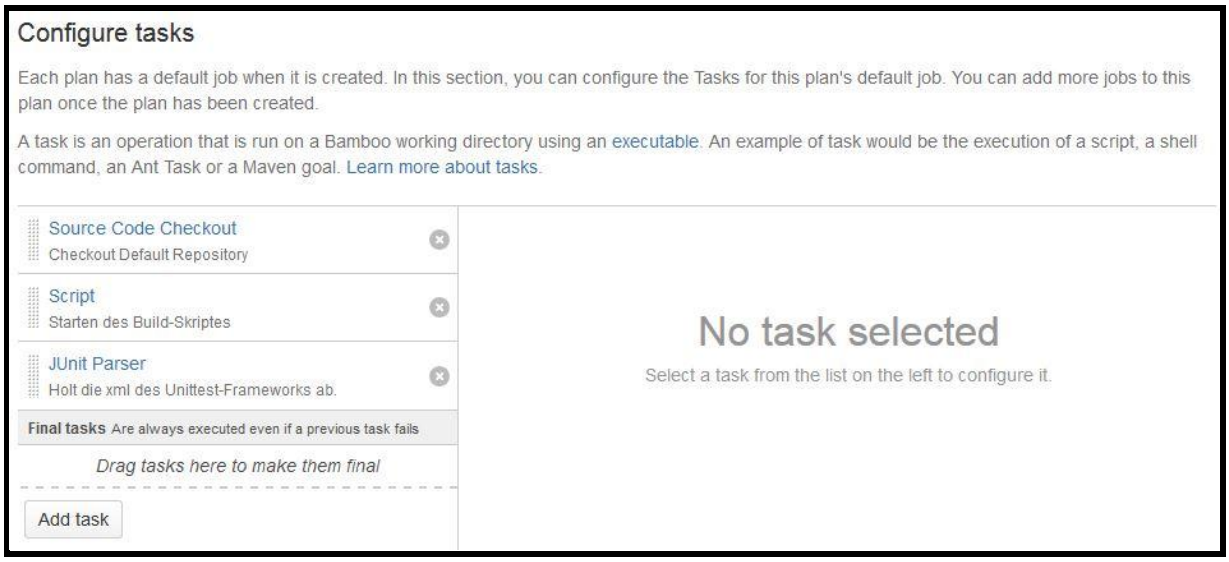

Abbildung 13: Bamboo Plankonfiguration 3/4

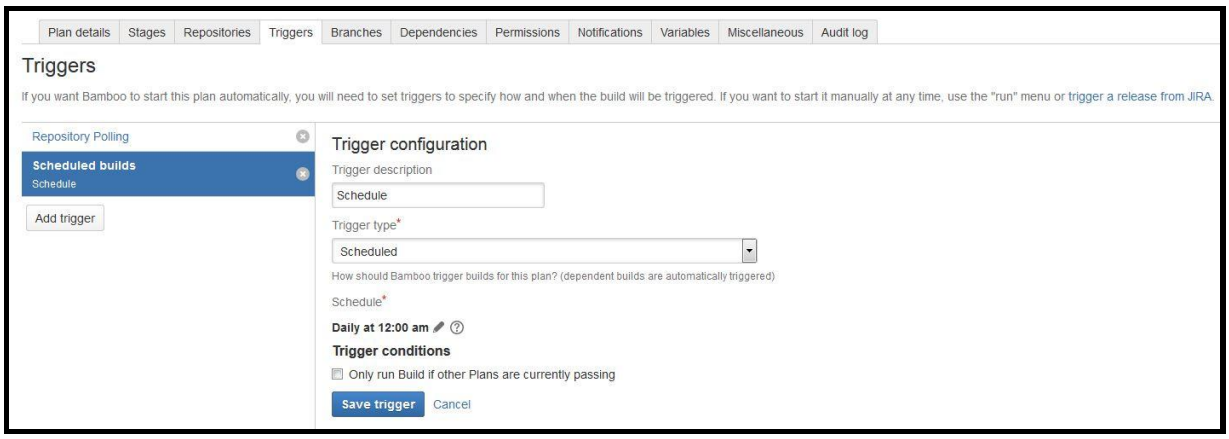

Abbildung 14: Bamboo Plankonfiguration 4/4

Tabelle 33: In Abbildung 11 bis 14 umgesetzte Anwendungsfälle und Anforderungen bei Bamboo

Zu Abbildung 11 bis 14	
Einstellung	**Anwendungsfall / Anforderung**
Script Configuration	Anwendungsfall 1
Configure tasks – JUnit Parser	Anwendungsfall 3
Triggers – Repository Polling	L00021-23-6
Triggers – Scheduled Builds	L00021-23-3

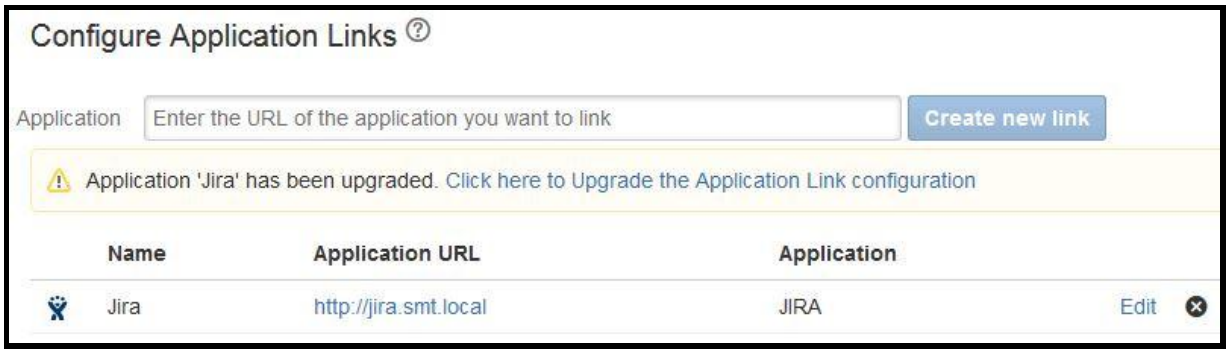

Abbildung 15: Bamboo-Jira-Konfiguration

Tabelle 34: In Abbildung 15 umgesetzte Anwendungsfälle und Anforderungen bei Bamboo

Zu Abbildung 15	
Einstellung	**Anwendungsfall / Anforderung**
Configure Application Links	L00021-23-13

Tabelle 35: Umsetzung der übrigen Anwendungsfälle und Anforderungen bei Bamboo

Anwendungsfall / Anforderung	**Erfüllt / nicht erfüllt**	**Begründung**
Anwendungsfall 2	Nicht erfüllt	Lint lässt sich nicht in Bamboo integrieren.
Anwendungsfall 4	Ausstehend	Die Interaktion mit den Schnittstellen des zu testenden Systems geschieht skriptbasiert. Bamboo unterstützt die Ausführung von Skripten, mit Hilfe derer die zu testenden Systeme angesprochen werden könnten. Etwaige Testergebnisse müssten von Bamboo entgegen genommen werden – beispielsweise über die Konsole oder über eine XML-Datei, die zu erzeugen wäre.
L00021-23-1	Erfüllt	Die Zuordnung von Anwendungsfällen zu Jobs wurde überprüft.
L00021-23-2	Erfüllt	Siehe hierzu *Bamboo-Lizenzmodell* in Tabelle 11.
L00021-23-4	Erfüllt	Neben den üblichen Auslösern wie *scheduled* und *poll repository* besteht auch die Möglichkeit, einen *release* aus Jira heraus auszulösen.
L00021-23-5	Nicht erfüllt	Eine Integration von Polarion in Bamboo ist zu vertretbaren Kosten nicht möglich.
L00021-23-7	Nicht erfüllt	Die Installation des *Atlassian Connector for Eclipse*[7] ist fehlerhaft. Das Plug-In lässt sich nicht verwenden.
L00021-23-8	Erfüllt	Bamboo bietet Möglichkeiten, aktuelle Job-Ergebnisse zu präsentieren.
L00021-23-9	Erfüllt	Bamboo bietet Möglichkeiten, in der Vergangenheit erzeugte Job-Ergebnisse zu präsentieren.
L00021-23-10	Nicht erfüllt	Es ist weder in Bamboo direkt noch durch ein Plug-In vorgesehen, Job-Ergebnisse in einem zu Microsoft Office 2010 kompatiblen Format zu dokumentieren.
L00021-23-11	Nicht erfüllt	Es ist weder in Bamboo direkt noch durch ein Plug-In vorgesehen, Job-Ergebnisse im PDF-Format zu dokumentieren.
L00021-23-12	Ausstehend	Diese Anforderung muss bewertet werden, sobald Polarion in der SMT-Softwareentwicklung eingesetzt wird.
L00021-24-1	Erfüllt	Die Bamboo-Oberfläche wurde sowohl in Firefox als auch im Internet Explorer 10 aufgerufen und geprüft.
L00021-24-2	Erfüllt	Die Bamboo-Oberfläche wurde sowohl in Firefox als auch im Internet Explorer 10 aufgerufen und geprüft.
L00021-24-3	Nicht anwendbar	Diese Anforderung lässt sich nicht überprüfen, da kein Testserver zur Verfügung steht.
L00021-24-4	Nicht anwendbar	Diese Anforderung lässt sich nicht überprüfen, da kein Testserver zur Verfügung steht.
L00021-24-5	Erfüllt	Siehe hierzu *Bamboo-Lizenzmodell* in Tabelle 11.

[7] Getestet auf Eclipse mit der Version Indigo Service Release 2 (Build id: 20120216-1857)

7.6 Team City

Für die Installation von Team City steht ebenfalls eine *setup.exe* zur Verfügung. Während der Installation besteht die Möglichkeit, Team City als Windows-Dienst einzurichten.

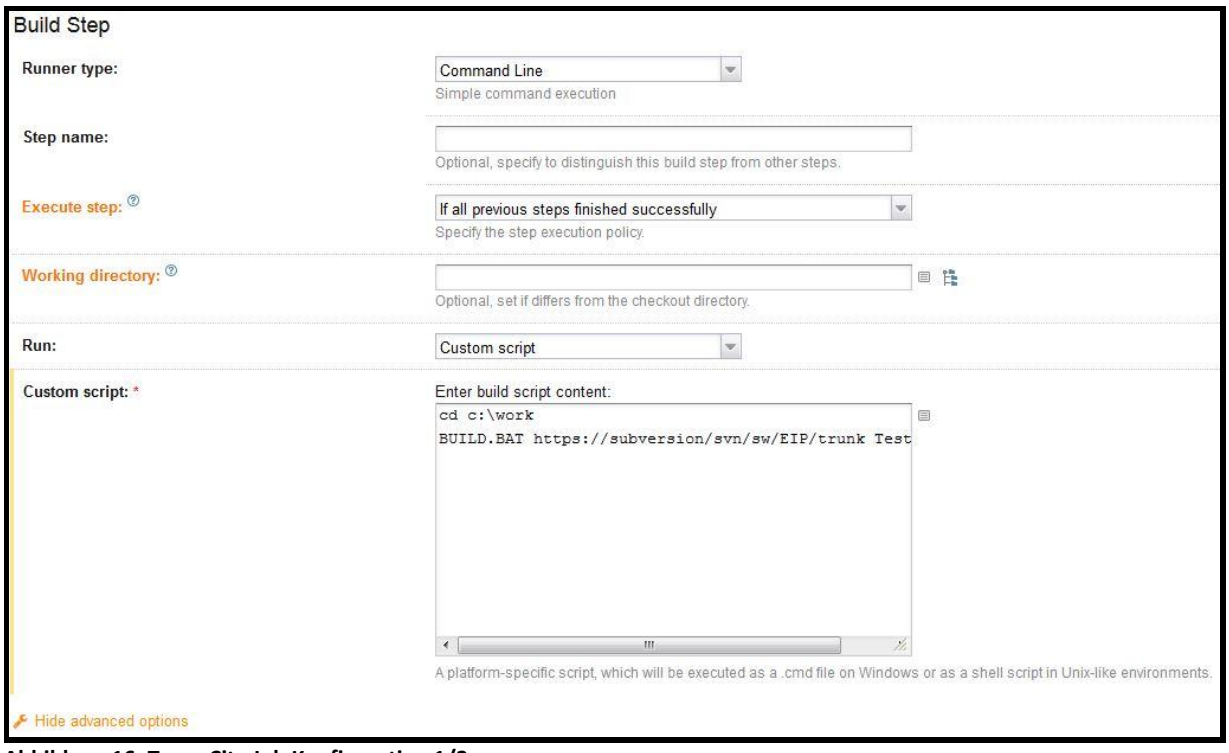

Abbildung 16: Team City Job Konfiguration 1/2

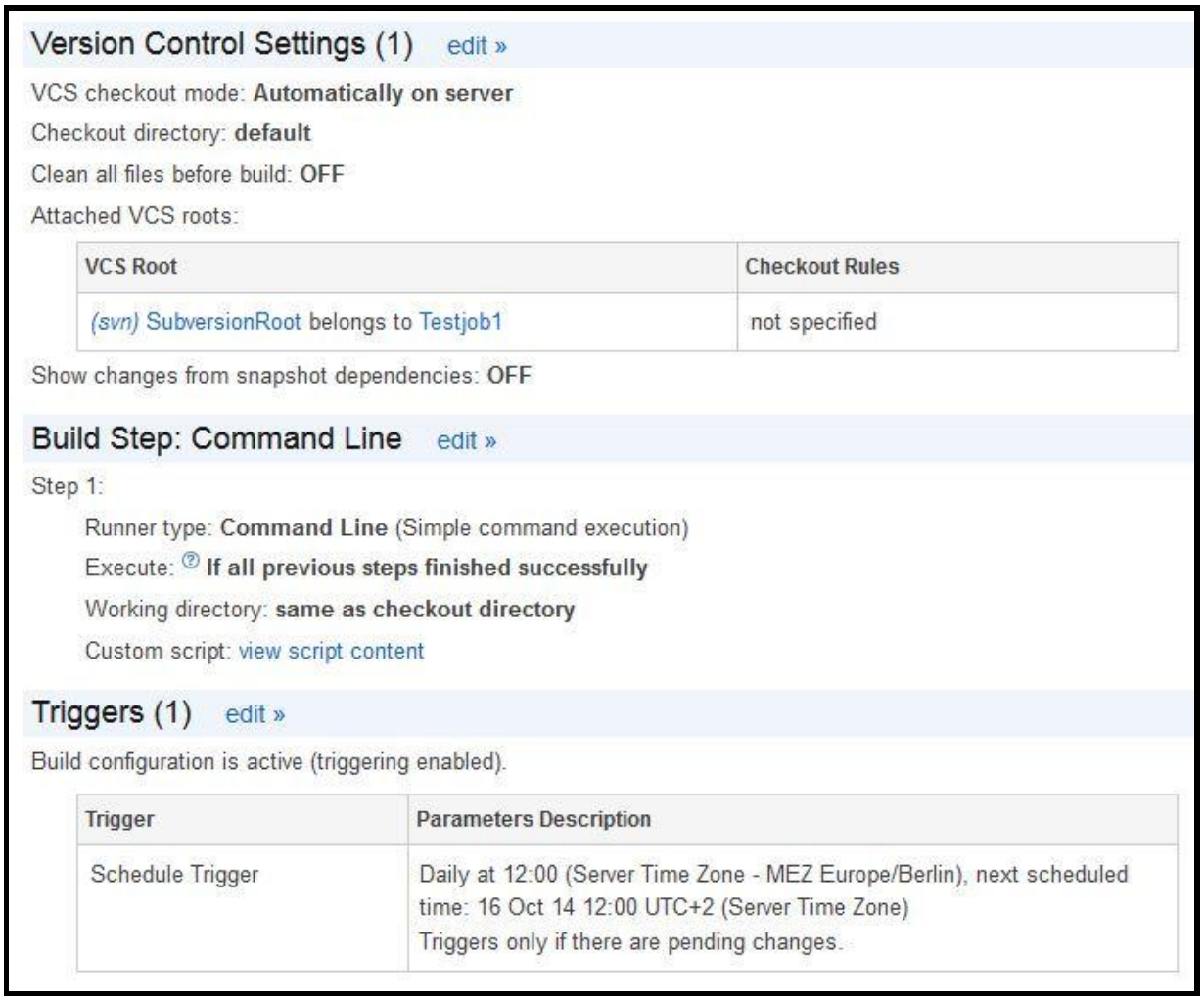

Abbildung 17: Team City Job Konfiguration 2/2

Tabelle 36: In Abbildung 16 und 17 umgesetzte Anwendungsfälle und Anforderungen bei Team City

Zu Abbildung 16 und 17	
Einstellung	**Anwendungsfall / Anforderung**
Build Step – Command Line	Anwendungsfall 1
Triggers – Schedule Trigger	L00021-23-3

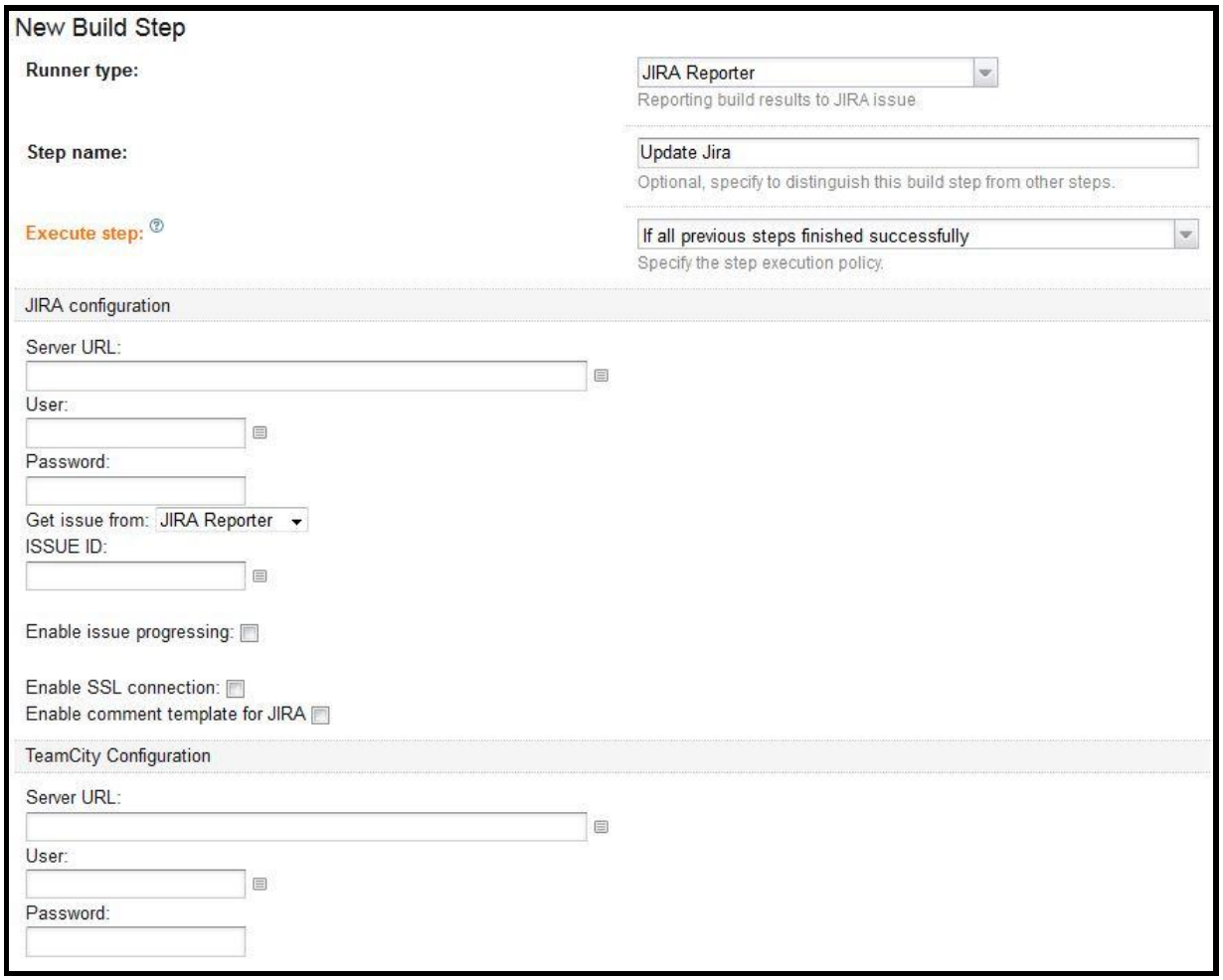

Abbildung 18: Team City-Jira-Konfiguration

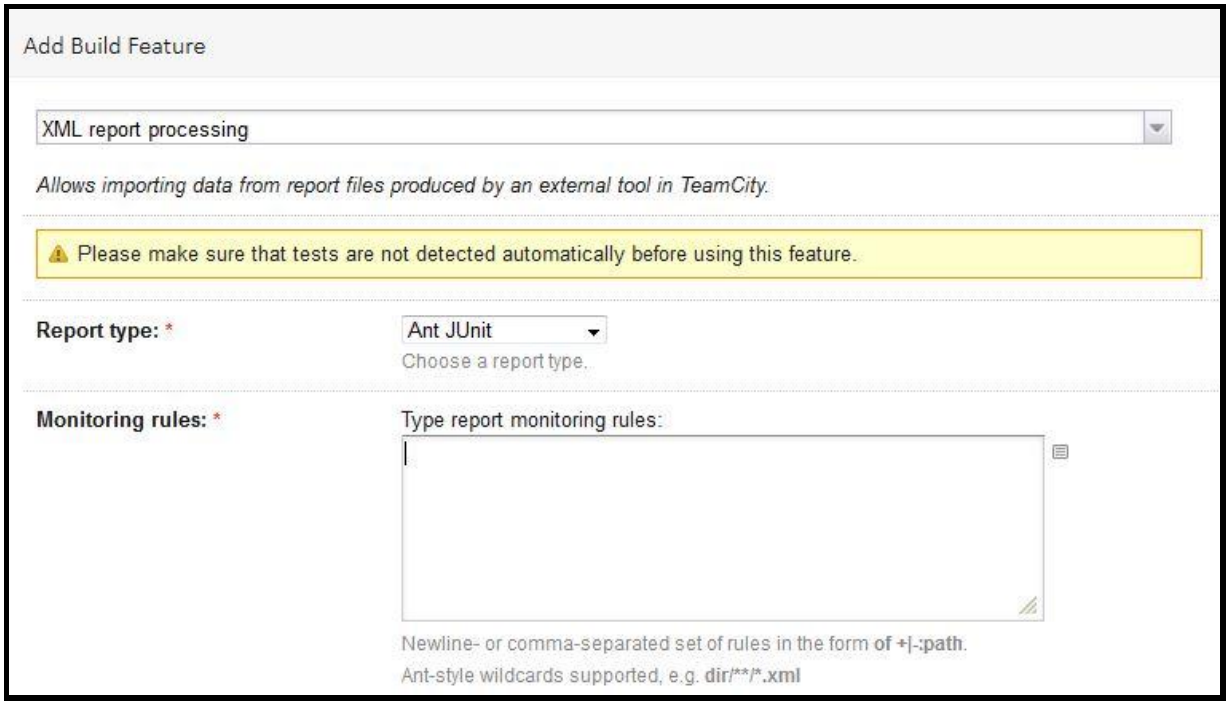

Abbildung 19: Team City-JUnit-Konfiguration

Tabelle 37: In Abbildung 18 umgesetzte Anwendungsfälle und Anforderungen bei Team City

Zu Abbildung 18 und 19	
Einstellung	**Anwendungsfall / Anforderung**
New Build Step – Runner type: Jira Reporter	L00021-23-13
Add Build Feature – XML report processing	Anwendungsfall 3

Tabelle 38: Umsetzung der übrigen Anwendungsfälle und Anforderungen bei Team City

Anwendungsfall / Anforderung	**Erfüllt / nicht erfüllt**	**Begründung**
Anwendungsfall 2	Nicht erfüllt	Lint lässt sich nicht in Team City integrieren.
Anwendungsfall 4	Ausstehend	Die Interaktion mit den Schnittstellen des zu testenden Systems geschieht skriptbasiert. Team City unterstützt die Ausführung von Skripten, mit Hilfe derer die zu testenden Systeme angesprochen werden könnten. Etwaige Testergebnisse müssten von Team City entgegen genommen werden – beispielsweise über die Konsole oder über eine XML-Datei, die zu erzeugen wäre.
L00021-23-1	Erfüllt	Die Zuordnung von Anwendungsfällen zu Jobs wurde überprüft.
L00021-23-2	Erfüllt	Siehe hierzu *Team City Lizenzmodell* in Tabelle 20.
L00021-23-4	Erfüllt	Neben den üblichen Auslösern wie *scheduled* und *repository* gibt es auch die Möglichkeit, nach vorangegangenen erfolgreichen Builds zu builden oder einen neuen Buildvorgang zu versuchen.
L00021-23-5	Nicht erfüllt	Die Verknüpfung mit Polarion ist weder direkt durch Cruise Control noch durch ein Plug-In vorgesehen.
L00021-23-6	Erfüllt	Der Auslöser *repository* ist vorhanden.
L00021-23-7	Nicht erfüllt	Das Plug-In funktioniert auf der getesteten Eclipse-Version[8] nicht wie beschreiben.
L00021-23-8	Erfüllt	Team City bietet Möglichkeiten, aktuelle Job-Ergebnisse zu präsentieren.
L00021-23-9	Erfüllt	Team City bietet Möglichkeiten, in der Vergangenheit erzeugte Job-Ergebnisse zu präsentieren.
L00021-23-10	Nicht erfüllt	Es ist weder in Team City direkt noch durch ein Plug-In vorgesehen, Job-Ergebnisse in einem zu Microsoft Office 2010 kompatiblen Format zu dokumentieren.
L00021-23-11	Nicht erfüllt	Es ist weder in Team City direkt noch durch ein Plug-In vorgesehen, Job-Ergebnisse im PDF-Format zu dokumentieren.
L00021-23-12	Ausstehend	Diese Anforderung muss bewertet werden, sobald Polarion in der SMT-Softwareentwicklung eingesetzt wird.
L00021-24-1	Erfüllt	Die Team City-Oberfläche wurde sowohl in Firefox als auch im Internet Explorer 10 aufgerufen und geprüft.
L00021-24-2	Erfüllt	Die Team City-Oberfläche wurde sowohl in Firefox als auch im Internet Explorer 10 aufgerufen und geprüft.
L00021-24-3	Nicht anwendbar	Diese Anforderung lässt sich nicht überprüfen, da kein Testserver zur Verfügung steht.

[8] Getestet auf Eclipse mit der Version Indigo Service Release 2 (Build id: 20120216-1857)

L00021-24-4	Nicht anwendbar	Diese Anforderung lässt sich nicht überprüfen, da kein Testserver zur Verfügung steht.
L00021-24-5	Erfüllt	Siehe hierzu *Team City Lizenzmodell* in Tabelle 20.

7.7 Auswertung

Alle getesteten *Build-Server* lassen sich einfach handhaben. Die grundlegenden Ansätze bei der Menüführung sind dieselben, lediglich kleinere Unterschiede bei einzelnen Begriffen – wie etwa *Job* oder *Plan* – oder der Struktur bei der Erstellung eines neuen Projekts lassen sich feststellen. Hier hat Bamboo den ansprechendsten Ansatz – dessen Oberfläche ist am übersichtlichsten.
Dennoch bieten Jenkins und Hudson den größeren Funktionsumfang. Die Integration von Lint sowie Polarion ist nur hier möglich.
Im Folgenden soll noch einmal aufgezeigt werden, wie viele der Anforderungen die getesteten *Build-Server* tatsächlich erfüllen können. Die Anforderungen 24-3 und 24-4 werden bei allen als erfüllt angenommen.

Tabelle 39: Erfüllte Anforderungen bei den getesteten *Build-Servern*

	Hudson	Jenkins	Bamboo	Team City
Erfüllte Anforderungen	77,3 %	77,3 %	68,2 %	68,2 %

Da insbesondere die Benutzung von Lint für den SMT-Entwicklungsprozess eine große Relevanz hat, kommen Bamboo und Team City für diesen nicht mehr in Frage. Bei einer Entscheidung zwischen Hudson und Jenkins müssen mehrere Punkte in Betracht gezogen werden:

- Hudson hat größere Release-Zyklen; die Releases sind auf eine längerfristige Benutzung der jeweiligen Version ausgelegt (*long term support).* Auch Jenkins hat mittlerweile die Wichtigkeit von längerfristigem Support einer Version erkannt und bietet deshalb die Wahl zwischen kleineren Release-Zyklen (*latest and greatest)* und *long term support.*
- Jenkins bietet eine größere Auswahl an Plug-Ins; die für den SMT-Entwicklungsprozess relevanten Plug-Ins sind allerdings auch für Hudson verfügbar.
- Die Handhabung, das heißt die Installation und die Wartung ist bei Jenkins etwas einfacher. Zur Wartung gehört beispielsweise die Installation sowie das Updaten und Verwalten von Plug-Ins.

Aufgrund dieser – wenn auch geringen – Unterschiede kann an dieser Stelle eine Empfehlung für Jenkins ausgesprochen werden.

8 Fazit und Ausblick

8.1 Fazit

Die auf den ersten Blick zahlreichen Möglichkeiten für die Einführung einer Kontinuierlichen Integration nehmen bei schrittweise genauerer Betrachtung angesichts der speziellen Anforderungen, die sich aus der SMT-Entwicklungsumgebung ableiten, immer weiter ab. Nachdem im ersten Schritt die Entscheidung getroffen wurde, sich bei der Suche einer möglichen Lösung auf *Build-Server* zu konzentrieren, wurden sechs der aussichtsreichsten mit Hilfe von Recherchen auf eine mögliche Erfüllung der gegebenen Anforderungen hin untersucht. Diese qualitative Bewertung, die sich ausschließlich auf Rechercheergebnisse stützte, wurde anschließend mit Hilfe eines Punktwertsystems quantifiziert. Hierbei konnte die Auswahl auf vier *Build-Server* eingegrenzt werden – diese lagen bei der Punktebewertung eng beieinander und wurden im Rahmen einer Testinstallation weiter bewertet.

Die Handhabung ist grundsätzlich bei allen *Build-Servern* ähnlich – neue Projekte lassen sich intuitiv erstellen, die auszuführenden Schritte können definiert und in gewünschter zeitlicher Abfolge angeordnet werden. Betrachtet man allein die Grundfunktionalität eines *Build-Servers*, schneiden die vier getesteten nahezu identisch ab.
Einige speziellere Anforderungen wie die Integration des Softwareanalysetools *Lint* sowie *Polarion* lassen sich nur mit den *Build-Servern* Hudson und Jenkins realisieren. Da dies der einzige Unterschied bei der Funktionalität ist, schneiden die beiden kostenlosen Varianten, was den reinen Funktionsumfang betrifft, sogar besser ab als die beiden kostenpflichtigen. Die somit auf Hudson und Jenkins reduzierte Wahl fällt aufgrund einiger leichter Vorteile – wie zum Beispiel einer etwas einfacheren Handhabung – zu Gunsten von Jenkins aus.

Dennoch lassen sich einige Anforderungen nicht umsetzen. Hierzu zählt der Export von Testergebnissen in einem Microsoft-Office kompatiblen oder PDF-Format. Auch die verschiedenen Eclipse-Plug-Ins der jeweiligen *Build-Server* lassen sich nicht korrekt auf der verwendeten Eclipse-Version[9] installieren.

[9] Eclipse mit der Version Indigo Service Release 2 (Build id: 20120216-1857)

8.2 Ausblick

Das grundlegende Aufsetzen eines CI-Servers gestaltet sich weitgehend unproblematisch – die hierfür notwendigen Voraussetzungen sind bereites gegeben. Um jedoch tatsächlich den vollständigen Entwicklungsprozess auf einen CI-Server zu verlagern, sind nach wie vor einige weitergehende Voraussetzungen zu schaffen sowie einige Dinge zu beachten.
Bisher wurde nur auf einer lokalen Workstation installiert. Die Installation auf einem Server steht noch aus – diese dürfte sich allerdings kaum von einer Installation auf einer lokalen Maschine unterscheiden.
Eine Automatisierung der Systemtests, die die letzte Ausbaustufe darstellen würde, erfordert die Zusammenarbeit einiger Systeme wie der Versionsverwaltung, dem System zur Interaktion des zu testenden Systems, dem Dateisystem und schließlich dem zu testenden System selbst. Die Interaktion mit den Schnittstellen des zu testenden Systems findet skriptbasiert statt – hierfür kommt das Jenkins *PostBuildScript Plugin*[10] in Frage. Dieses erlaubt die Ausführung von verschiedenen Skripten wie beispielsweise einem *Windows batch file* im Anschluss an einen Buildvorgang. Es bliebe zu überprüfen, wie etwaige Testresultate von Jenkins entgegen genommen werden könnten.
Schließlich bleibt zu klären, wie sich der Einsatz eines CI-Servers mit der für Softwareentwicklung in der Medizintechnik relevanten Norm DIN EN 62304:2007-03 in Einklang bringen lässt. Diese fordert beispielsweise vom Hersteller „das Verfahren und die Umgebung (zu) dokumentieren, die verwendet wurden, um die freigegebene Software zu erzeugen“ [111, Abschnitt 5.8.5].

[10] https://wiki.jenkins-ci.org/display/JENKINS/PostBuildScript+Plugin (abgerufen am 23. Oktober 2014)

Quellenverzeichnis

	Quelle	Anmerkungen	Abrufdatum
[1]	http://wiki.eclipse.org/Building_a_software_project		02.09.2014
[2]	http://wiki.eclipse.org/Building_a_software_project #Builds_by_changes_in_Subversion.2FCVS		02.09.2014
[3]	http://wiki.eclipse.org/Building_a_software_project #Builds_by_changes_in_Subversion.2FCVS		02.09.2014
[4]	http://hudson-ci.org/PluginCentral/site/3.2/		02.09.2014
[5]	http://extensions.polarion.com/extensions/97-polarion-integration-plugin-for-hudson		02.09.2014
[6]	http://wiki.eclipse.org/Building_a_software_project #Builds_by_changes_in_Subversion.2FCVS		02.09.2014
[7]	http://marketplace.eclipse.org/content/hudsonjenkins-mylyn-builds-connector#.VAakc2OHiAI		03.09.2014
[8]	http://wiki.eclipse.org/Starting_and_Accessing_Hudson		03.09.2014
[9]	http://wiki.eclipse.org/Starting_and_Accessing_Hudson		03.09.2014
[10]	http://hudson-ci.org/PluginCentral/	Stichwort *Polarion*	03.09.2014
[11]	https://marketplace.atlassian.com/plugins/com.marvelution.jira.plugins.jenkins		03.09.2014
[12]	http://hudson-ci.org/PluginCentral/	Stichwort *Jira*	03.09.2014
[13]	http://wiki.eclipse.org/Hudson-ci/Meet_Hudson		03.09.2014
[14]	http://wiki.eclipse.org/Hudson-ci/Meet_Hudson		03.09.2014
[15]	http://wiki.eclipse.org/Hudson-ci/Installing_Hudson		03.09.2014
[16]	http://wiki.eclipse.org/Hudson-ci/Installing_Hudson		03.09.2014
[17]	http://www.eclipse.org/legal/epl/notice.php		03.09.2014
[18]	https://github.com/jenkinsci/jenkins/blob/master/LICENSE.txt		03.09.2014
[19]	https://wiki.jenkins-ci.org/display/JENKINS/Building+a+software+project		03.09.2014
[20]	https://wiki.jenkins-ci.org/display/JENKINS/Building+a+software+project	*Configuring automatic builds*	03.09.2014
[21]	https://wiki.jenkins-ci.org/display/JENKINS/Building+a+software+project	*Configuring automatic builds*	03.09.2014
[22]	https://wiki.jenkins-ci.org/display/JENKINS/Polarion+Plugin		03.09.2014
[23]	https://wiki.jenkins-ci.org/display/JENKINS/Building+a+software+project	*Using a post-commit trigger in CVS*	03.09.2014
[24]	http://marketplace.eclipse.org/content/hudsonjenkins-mylyn-builds-connector#.VAakc2OHiAI		03.09.2014
[25]	https://wiki.jenkins-ci.org/display/JENKINS/Starting+and+Accessing+Jenk	*Accessing Jenkins*	03.09.2014

	Quelle	Anmerkungen	Abrufdatum
	ins#StartingandAccessingJenkins-top		
[26]	https://wiki.jenkins-ci.org/display/JENKINS/Starting+and+Accessing+Jenkins#StartingandAccessingJenkins-top	*Accessing Jenkins*	03.09.2014
[27]	https://wiki.jenkins-ci.org/display/JENKINS/Polarion+Plugin		03.09.2014
[28]	https://marketplace.atlassian.com/plugins/com.marvelution.jira.plugins.jenkins		03.09.2014
[29]	https://wiki.jenkins-ci.org/display/JENKINS/JIRA+Plugin		03.09.2014
[30]	https://wiki.jenkins-ci.org/display/JENKINS/Starting+and+Accessing+Jenkins#StartingandAccessingJenkins-top		03.09.2014
[31]	https://wiki.jenkins-ci.org/display/JENKINS/Starting+and+Accessing+Jenkins#StartingandAccessingJenkins-top	*Accessing Jenkins*	03.09.2014
[32]	http://jenkins-ci.org/		03.09.2014
[33]	http://jenkins-ci.org/		03.09.2014
[34]	https://github.com/jenkinsci/jenkins/blob/master/LICENSE.txt		03.09.2014
[35]	https://wiki.jenkins-ci.org/display/JENKINS/Plugins		21.08.2014
[36]	http://hudson-ci.org/PluginCentral/		22.08.2014
[37]	https://wiki.jenkins-ci.org/display/JENKINS/LTS+Release+Line		21.08.2014
[38]	https://confluence.atlassian.com/display/BAMBOO/Understanding+the+Bamboo+CI+Server		03.09.2014
[39]	https://www.atlassian.com/software/bamboo/pricing		03.09.2014
[40]	https://confluence.atlassian.com/display/BAMBOO/triggering	Stichwort *Cron-based scheduling*	03.09.2014
[41]	https://confluence.atlassian.com/display/BAMBOO/triggering		03.09.2014
[42]	http://blog.polarion.com/gui-test-automation-with-ranorex-and-polarion/		25.08.2014
[43]	https://confluence.atlassian.com/display/BAMBOO/Repository+triggers+the+build+when+changes+are+committed		03.09.2014
[44]	http://marketplace.eclipse.org/content/atlassian-connector-eclipse		03.09.2014
[45]	http://marketplace.eclipse.org/content/atlassian-connector-eclipse		03.09.2014
[46]	http://blog.polarion.com/gui-test-automation-with-ranorex-and-polarion/		25.08.2014
[47]	https://confluence.atlassian.com/display/BAMBOO/Integrating+Bamboo+with+JIRA		03.09.2014
[48]	https://confluence.atlassian.com/display/BAMBOO/Configuring+plans		03.09.2014
[49]	https://www.atlassian.com/software/bamboo		03.09.2014
[50]	https://confluence.atlassian.com/display/BAMBOO/Installing+Bamboo+on+Windows		03.09.2014

	Quelle	Anmerkungen	Abrufdatum
[51]	https://confluence.atlassian.com/display/BAMBOO/Installing+Bamboo+on+Windows		03.09.2014
[52]	https://www.atlassian.com/software/bamboo/pricing		03.09.2014
[53]	http://msdn.microsoft.com/en-us/library/dd492668.aspx		25.08.2014
[54]	http://sourceforge.net/projects/cruisecontrol/files/CruiseControl/1.0/		28.08.2014
[55]	http://cruisecontrol.sourceforge.net/download.html		28.08.2014
[56]	http://cruisecontrol.sourceforge.net/overview.html		29.08.2014
[57]	http://cruisecontrol.sourceforge.net/main/configxml.html#tfs	Abschnitt *Schedule*	29.08.2014
[58]	http://cruisecontrol.sourceforge.net/main/configxml.html	Stichwort *Trigger*	29.08.2014
[59]	http://cruisecontrol.sourceforge.net/main/configxml.html	Stichwort *SVN*	29.08.2014
[60]	http://cruiseedit.sourceforge.net/		29.08.2014
[61]	http://cruisecontrol.sourceforge.net/dashboard.html		29.08.2014
[62]	http://cruisecontrol.sourceforge.net/dashboard.html		29.08.2014
[63]	https://marketplace.atlassian.com/search?q=cruise+control		29.08.2014
[64]	http://cruisecontrol.sourceforge.net/main/configxml.html		29.08.2014
[65]	http://cruisecontrol.sourceforge.net/dashboard.html		29.08.2014
[66]	http://cruisecontrol.sourceforge.net/gettingstartedbindist.html		29.08.2014
[67]	http://cruisecontrol.sourceforge.net/gettingstartedbindist.html		29.08.2014
[68]	http://cruisecontrol.sourceforge.net/license.html		29.08.2014
[69]	http://confluence.jetbrains.com/display/TCD8/Command+Line		02.09.2014
[70]	http://confluence.jetbrains.com/display/TW/TeamCity+Plugins?_ga=1.174735147.823790237.1406207757		01.09.2014
[71]	http://www.jetbrains.com/teamcity/features/	Stichwort *Projects Hierarchy*	01.09.2014
[72]	http://confluence.jetbrains.com/display/TCD8/Configuring+Schedule+Triggers		01.09.2014
[73]	http://confluence.jetbrains.com/display/TCD8/Configuring+Schedule+Triggers		01.09.2014
[74]	http://confluence.jetbrains.com/display/TCD8/Subversion		01.09.2014
[75]	http://confluence.jetbrains.com/display/TCD8/Eclipse+Plugin		01.09.2014
[76]	http://www.jetbrains.com/teamcity/features/#Continuous_Integration		02.09.2014
[77]	http://www.jetbrains.com/teamcity/features/#Conti		02.09.2014

	Quelle	Anmerkungen	Abrufdatum
	nuous_Integration		
[78]	https://github.com/mamirov/jirareporter/wiki/TeamCity-JIRA-Reporter-%28ENG%29		02.09.2014
[79]	http://www.jetbrains.com/teamcity/features/#Build_Configuration		02.09.2014
[80]	http://www.jetbrains.com/teamcity/features/#Continuous_Integration		02.09.2014
[81]	http://confluence.jetbrains.com/display/TCD8/Installation		02.09.2014
[82]	http://confluence.jetbrains.com/display/TCD8/How+To...#HowTo...-hardwarerequirements	Stichwort *Windows Server 2008*	02.09.2014
[83]	http://confluence.jetbrains.com/display/TCD8/Installation		02.09.2014
[84]	http://confluence.jetbrains.com/display/TCD8/How+To...#HowTo...-hardwarerequirements	Stichwort *Windows Server 2008*	02.09.2014
[85]	http://www.jetbrains.com/teamcity/buy/		02.09.2014
[86]	http://msdn.microsoft.com/en-us/magazine/cc188695.aspx		05.09.2014
[87]	http://msdn.microsoft.com/de-de/library/bb668975.aspx		05.09.2014
[88]	http://msdn.microsoft.com/en-us/library/ms181286.aspx		05.09.2014
[89]	https://svnbridge.codeplex.com/		05.09.2014
[90]	http://msdn.microsoft.com/de-de/library/hh913026.aspx		08.09.2014
[91]	http://msdn.microsoft.com/en-us/library/vstudio/ms181721.aspx		08.09.2014
[92]	https://marketplace.atlassian.com/plugins/com.spartez.jira.plugins.bork.tfs4jira		08.09.2014
[93]	http://msdn.microsoft.com/en-us/library/vstudio/ms181725.aspx		08.09.2014
[94]	http://msdn.microsoft.com/de-de/library/bb892990%28v=vs.90%29.aspx		09.09.2014
[95]	http://msdn.microsoft.com/en-us/library/dd578592.aspx		09.09.2014
[96]	http://www.microsoftstore.com/store/msde/de_DE/pdp/Visual-Studio-Team-Foundation-Server-2013/productID.288673900		09.09.2014
[97]	http://www.eclipse.org/hudson/the-hudson-book/book-hudson.chunked/ch06.html		09.09.2014
[98]	https://wiki.jenkins-ci.org/display/JENKINS/Building+a+software+project		09.09.2014
[99]	https://confluence.atlassian.com/display/BAMBOO/Defining+a+new+executable+capability		09.09.2014
[100]	http://msdn.microsoft.com/en-us/library/dd393574.aspx		10.09.2014
[101]	http://cruisecontrol.sourceforge.net/main/configxml.html#exec-examples	Stichwort *<Exec>*	10.09.2014
[102]	http://docs.oracle.com/cd/E18941_01/tutorials/jdtut_11r2_98/jdtut_11r2_98_3.html	Stichwort *JUnit*	10.09.2014

	Quelle	Anmerkungen	Abrufdatum
[103]	https://wiki.jenkins-ci.org/display/JENKINS/Unit+Test		11.09.2014
[104]	https://confluence.atlassian.com/display/BAMBOO/JUnit+Parser		11.09.2014
[105]	http://visualstudiogallery.msdn.microsoft.com/2011f516-15a7-4f9a-8b86-1e0894a75739		11.09.2014
[106]	http://cruisecontrol.sourceforge.net/gettingstartedsourcedist.html	Stichwort *unittests.xsl*	11.09.2014
[107]	http://www.jetbrains.com/teamcity/features/	Stichwort *Technology Awarness*	12.09.2014
[108]	http://www.jetbrains.com/idea/webhelp/run-debug-configuration-junit.html		12.09.2014
[109]	http://wiki.hudson-ci.org/display/HUDSON/Warnings+Plugin;jsessionid=1CD9FDA385F3AE44C75C0010704FAB81		26.09.2014
[110]	https://wiki.jenkins-ci.org/display/JENKINS/Warnings+Plugin		26.09.2014
[111]	DIN EN 62304:2007-03; VDE 0750-101:2007-03 Medizingeräte-Software – Software-Lebenszyklus-Prozesse		

Anhang

A Berechnungstabellen für die quantitative Bewertung

Tabelle A1: Punkteverteilung bei Gewichtung 1

Relevanz	Gewichtung	Umsetzbarkeit	Gewichtung
hoch (z.B. "muss")	3	einfach ("out of the box")	3
mittel (z.B. "soll")	2	mittel (z.B. Plug-In)	2
niedrig (z.B "kann")	1	schwierig (nur mit Aufwand)	1
		nicht herauszufinden vor Testinstallation	1
		nicht umsetzbar	0

Tabelle A2: Punkteverteilung bei Gewichtung 2

Relevanz	Gewichtung	Umsetzbarkeit	Gewichtung
hoch (z.B. "muss")	3	einfach ("out of the box")	3
mittel (z.B. "soll")	2	mittel (z.B. Plug-In)	2
niedrig (z.B "kann")	1	schwierig (nur mit Aufwand)	1
		nicht herauszufinden vor Testinstallation	0
		nicht umsetzbar	0

Tabelle A3: Punkteverteilung bei Gewichtung 3

Relevanz	Gewichtung	Umsetzbarkeit	Gewichtung
hoch (z.B. "muss")	3	einfach ("out of the box")	1
mittel (z.B. "soll")	2	mittel (z.B. Plug-In)	1
niedrig (z.B "kann")	1	schwierig (nur mit Aufwand)	1
		nicht herauszufinden vor Testinstallation	0
		nicht umsetzbar	0

Tabelle A4: Berechnung der Bewertung bei Gewichtung 1

Anforderung	Relevanz/ Wichtigkeit	Hudson		Jenkins		Bamboo		TFS		Cruise Control		Team City	
Anwendungsfälle													
Build	3	3		3		3		1		3		2	
stat. Analyse	3	2		2		2		1		1		1	
Unit Test	3	3		3		3		2		3		3	
Systemtest	3	1		1		1		1		1		1	
		27	75,00	27	75,00	27	75,00	15	41,67	24	66,67	21	58,333
funktionale Anforderungen													
L00021-23-1	3	3		3		3		3		3		3	
L00021-23-2	2	3		3		2		3		3		2	
L00021-23-3	3	3		3		3		3		3		3	
L00021-23-4	3	3		3		3		3		3		3	
L00021-23-5	1	1		1		1		0		0		0	
L00021-23-6	2	3		3		3		2		3		3	
L00021-23-7	2	2		2		2		2		0		2	
L00021-23-8	3	3		3		3		3		3		3	
L00021-23-9	3	3		3		3		3		3		3	
L00021-23-10	3	1		1		1		1		1		1	
L00021-23-11	2	1		1		1		1		1		1	
L00021-23-12	2	1		1		1		0		1		1	
L00021-23-13	3	2		2		3		1		1		2	
		75	78,13	75	78,13	76	79,17	67	69,79	67	69,79	72	75,00
nicht funktionale Anforderungen													
L00021-24-1	2	3		3		3		2		0		3	
L00021-24-2	2	3		3		3		2		3		3	
L00021-24-3	3	3		3		3		3		3		3	
L00021-24-4	3	3		3		3		3		3		3	

L00021-24-5	3	3		3		2		2		3		2	
		39	100	39	100	36	92,31	32	82,051	33	84,62	36	92,308
		141	82,46	141	82,46	139	81,29	114	66,67	124	72,51	129	75,44

Tabelle A5: Berechnung der Bewertung bei Gewichtung 2

Anforderung	Relevanz/ Wichtigkeit	Hudson		Jenkins		Bamboo		TFS		Cruise Control		Team City	
Anwendungsfälle													
Build	3	3		3		3		0		3		2	
stat. Analyse	3	2		2		2		0		0		0	
Unit Test	3	3		3		3		2		3		3	
Systemtest	3	0		0		0		0		0		0	
		24	66,67	24	66,67	24	66,67	6	16,67	18	50,00	15	41,667
funktionale Anforderungen													
L00021-23-1	3	3		3		3		3		3		3	
L00021-23-2	2	3		3		2		3		3		2	
L00021-23-3	3	3		3		3		3		3		3	
L00021-23-4	3	3		3		3		3		3		3	
L00021-23-5	1	0		0		1		0		0		0	
L00021-23-6	2	3		3		3		2		3		3	
L00021-23-7	2	2		2		2		2		0		2	
L00021-23-8	3	3		3		3		3		3		3	
L00021-23-9	3	3		3		3		3		3		3	
L00021-23-10	3	0		0		0		0		0		0	
L00021-23-11	2	0		0		0		0		0		0	
L00021-23-12	2	0		0		0		0		0		0	

L00021-23-13	3	2		2		3		1		1		2	
		67	69,79	67	69,79	69	71,88	62	64,58	60	62,50	65	67,71
nicht funktionale Anforderungen													
L00021-24-1	2	3		3		3		2		0		3	
L00021-24-2	2	3		3		3		2		3		3	
L00021-24-3	3	3		3		3		3		3		3	
L00021-24-4	3	3		3		3		3		3		3	
L00021-24-5	3	3		3		2		2		3		2	
		39	100	39	100	36	92,31	32	82,051	33	84,62	36	92,308
		130	76,02	130	76,02	129	75,44	100	58,48	111	64,91	116	67,84

Tabelle A6: Berechnung der Bewertung bei Gewichtung 3

Anforderung	Relevanz/ Wichtigkeit	Hudson		Jenkins		Bamboo		TFS		Cruise Control		Team City	
Anwendungsfälle													
Build	3	1		1		1		0		1		1	
stat. Analyse	3	1		1		1		0		0		0	
Unit Test	3	1		1		1		1		1		1	
Systemtest	3	0		0		0		0		0		0	
		9	75,00	9	75,00	9	75,00	3	25,00	6	50,00	6	50
funktionale Anforderungen													
L00021-23-1	3	1		1		1		1		1		1	
L00021-23-2	2	1		1		1		1		1		1	
L00021-23-3	3	1		1		1		1		1		1	
L00021-23-4	3	1		1		1		1		1		1	
L00021-23-5	1	0		0		1		0		0		0	

L00021-23-6	2	1		1		1		1		1		1	
L00021-23-7	2	1		1		1		1		0		1	
L00021-23-8	3	1		1		1		1		1		1	
L00021-23-9	3	1		1		1		1		1		1	
L00021-23-10	3	0		0		0		0		0		0	
L00021-23-11	2	0		0		0		0		0		0	
L00021-23-12	2	0		0		0		0		0		0	
L00021-23-13	3	1		1		1		1		1		1	
		24	75,00	24	75,00	25	78,13	24	75,00	22	68,75	24	75,00
nicht funktionale Anforderungen													
L00021-24-1	2	1		1		1		1		0		1	
L00021-24-2	2	1		1		1		1		1		1	
L00021-24-3	3	1		1		1		1		1		1	
L00021-24-4	3	1		1		1		1		1		1	
L00021-24-5	3	1		1		1		1		1		1	
		13	100	13	100	13	100,00	13	100	11	84,62	13	100
		46	80,70	46	80,70	47	82,46	40	70,18	39	68,42	43	75,44

B Batchfile

```
@ECHO OFF
REM checkout code from %1 to %2 [using revision %3]
IF NOT _%1==_--help GOTO BDOIT
:BHELP
ECHO BUILD subversion-path working-copy-dir [subversion-revision]
ECHO    checkout code from 'subversion-path' to 'working-copy-dir' [using
ECHO    'subversion-revision'] and build IGOS SWKs and SWSs
ECHO        in the current directory any exiting sub-directory 'working-copy-dir'
ECHO        will be deleted
ECHO        The newly generated sub-directory 'working-copy-dir' will be populated
ECHO        with the 'subversion-path'.
ECHO        If no 'subversion-revision' is given, the HEAD revision will be used
ECHO        instead.
ECHO        Several Working Files will be generated:
ECHO            BCOMPARGS.TXT Compiler Parameters
ECHO            BLOGGING.TXT Logging all stdout and stderr output
ECHO            BARCHARGS.TXT Archiver Parameters
ECHO            BLINKARGS.TXT Linker Parameters
ECHO        Existing C:\msys\1.0\bin\sed and C:\msys\1.0\bin\sort.exe are
ECHO        mandatory.
ECHO        EXAMPLE: BUILD.BAT https://subversion/svn/sw/EIP/trunk toto 8576
GOTO :eof

:BDOIT
@ECHO ON
IF _%1==_ GOTO BHELP
IF _%2==_ GOTO BHELP
IF EXIST %2 RMDIR /S /Q %2
MKDIR %2
IF _%3==_ SET BREV=HEAD
IF NOT _%3==_ SET BREV=%3
svn co %1 -r %BREV% %2
cd %2

REM this is the 'root-cause'
if NOT DEFINED BROOT SET BROOT=%cd%
if NOT DEFINED BLOGGING SET BLOGGING=%BROOT%\BLOGGING.LOG

ECHO %DATE%-%TIME% BROOT=%BROOT% >> %BLOGGING%
ECHO %DATE%-%TIME% BLOGGING=%BLOGGING% >> %BLOGGING%

ECHO ++++++++++++++++++++ >> %BLOGGING%
ECHO ++++++++++++++++++++ >> %BLOGGING%
ECHO %DATE%-%TIME% %~dpf0 started >> %BLOGGING%

REM if not available, 'generate' tools
if NOT DEFINED BCOMPILER SET BCOMPILER=C:\EWARM\arm\bin\iccarm.exe
if NOT DEFINED BARCHIVER SET BARCHIVER=C:\EWARM\arm\bin\iarchive.exe
if NOT DEFINED BLINKER SET BLINKER=C:\EWARM\arm\bin\ilinkarm.exe
ECHO %DATE%-%TIME% BCOMPILER=%BCOMPILER% >> %BLOGGING%
ECHO %DATE%-%TIME% BARCHIVER=%BARCHIVER% >> %BLOGGING%
ECHO %DATE%-%TIME% BLINKER=%BLINKER% >> %BLOGGING%

REM generate parameter File for the compiler
REM all include directories
DEL BHEADERS.TXT
FOR /R %%I IN (*.h) DO ECHO -I%%~dpI >> BHEADERS.TXT
REM test-directories to be deleted from list
C:\msys\1.0\bin\sed '/test/d' BHEADERS.TXT > BHEADERS.TMP
DEL BHEADERS.TXT & RENAME BHEADERS.TMP BHEADERS.TXT
REM SWK000006_UI_Darstellung-directories to be deleted from list
C:\msys\1.0\bin\sed '/SWK000006/d' BHEADERS.TXT > BHEADERS.TMP
DEL BHEADERS.TXT & RENAME BHEADERS.TMP BHEADERS.TXT
REM SWK0000058_RTE_for_Silverlight-directories to be deleted from list
C:\msys\1.0\bin\sed '/SWK000058/d' BHEADERS.TXT > BHEADERS.TMP
DEL BHEADERS.TXT & RENAME BHEADERS.TMP BHEADERS.TXT
```

```
REM SWS000002_Basisprozessor_Software-directories to be deleted from list
C:\msys\1.0\bin\sed '/SWKS00002/d' BHEADERS.TXT > BHEADERS.TMP
DEL BHEADERS.TXT & RENAME BHEADERS.TMP BHEADERS.TXT

REM generate compiler parameters
ECHO --no_wrap_diagnostics  > BCOMPARGS.TXT
ECHO -I inc\\ >> BCOMPARGS.TXT
C:\msys\1.0\bin\sort.exe -u BHEADERS.TXT >> BCOMPARGS.TXT
ECHO -
IC:/EWARM/arm/examples/ST/STM32F2xx/STM32F2xx_StdPeriph_Lib/Libraries/CMSIS/Device/
ST/STM32F2xx/Include >> BCOMPARGS.TXT
ECHO -
IC:/EWARM/arm/examples/ST/STM32F2xx/STM32F2xx_StdPeriph_Lib/Libraries/STM32F2xx_Std
Periph_Driver/inc >> BCOMPARGS.TXT
ECHO -IC:/EWARM/arm/CMSIS/Include" >> BCOMPARGS.TXT
ECHO --eec++ >> BCOMPARGS.TXT
ECHO --cpu Cortex-M3 >> BCOMPARGS.TXT
ECHO --fpu None >> BCOMPARGS.TXT
ECHO --debug >> BCOMPARGS.TXT
ECHO --dlib_config C:\EWARM\arm\inc\c\DLib_Config_Normal.h >> BCOMPARGS.TXT
ECHO --endian little >> BCOMPARGS.TXT
ECHO --cpu_mode thumb >> BCOMPARGS.TXT
ECHO -On >> BCOMPARGS.TXT
ECHO --no_cse >> BCOMPARGS.TXT
ECHO --no_unroll >> BCOMPARGS.TXT
ECHO --no_inline >> BCOMPARGS.TXT
ECHO --no_code_motion >> BCOMPARGS.TXT
ECHO --no_tbaa >> BCOMPARGS.TXT
ECHO --no_clustering >> BCOMPARGS.TXT
ECHO --no_scheduling >> BCOMPARGS.TXT

DEL BHEADERS.TXT

REM call compiler for Common, SWK and SWS
for /D /R %%I IN (Common*) DO ECHO CALL %~dp0\BCOMPSWK.BAT %BROOT% %%I >>
%BLOGGING% && CALL %~dp0\BCOMPSWK.BAT %BROOT% %%I
for /D /R %%I IN (SWK\SWK*) DO ECHO CALL %~dp0\BCOMPSWK.BAT %BROOT% %%I >>
%BLOGGING% && CALL %~dp0\BCOMPSWK.BAT %BROOT% %%I
for /D /R %%I IN (SWS\SWS*) DO ECHO CALL %~dp0\BCOMPSWS.BAT %BROOT% %%I >>
%BLOGGING% && CALL %~dp0\BCOMPSWS.BAT %BROOT% %%I

CD ..

ECHO %DATE%-%TIME% %~dpf0 ended >> %BLOGGING%
ECHO ---------------------- >> %BLOGGING%
ECHO ---------------------- >> %BLOGGING%
```