Bachelorarbeit

Sascha Kluth

Einführung eines Open Source Build Management und Continuous Integration Systems in eine Java-Entwicklungsumgebung

Fakultät Technik und Informatik
Department Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Sascha Kluth

Einführung eines Open Source Build Management und Continuous Integration Systems in eine Java-Entwicklungsumgebung

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung
im Studiengang Angewandte Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Jörg Raasch
Zweitgutachter: Prof. Dr. Olaf Zukunft

Abgegeben am 1.Februar 2008

Sascha Kluth

Thema der Arbeit

Einführung eines Open Source Build Management und Continuous Integration Systems in eine Java-Entwicklungsumgebung

Stichworte

Build Management, Continuous Integration, Qualitätssicherung, Versionskontrolle, Test, integrierte Entwicklungsumgebung

Kurzzusammenfassung

Nach der Behandlung der zu Grunde liegenden Konzepte zum Build Management und der Continuous Integration werden Produkte recherchiert, mit deren Hilfe die Vision einer integrierten Build Management und Continuous Integration Unterstützung umgesetzt werden kann. Die Recherche ist dabei auf kostenfrei nutzbare (i.d.R. OpenSource-) Produkte beschränkt und beginnt mit den zur Verfügung stehenden Continuous Integration Produkten. Da diese noch nicht sämtliche Unterstützungsmöglichkeiten integrieren, sondern auf weitere Produkte für das Build Management, die Testumgebung und die Versionskontrolle angewiesen sind, werden diese Produkte ebenfalls auf ihre Kombinierbarkeit und ihren Funktionsumfang hin überprüft. Letztendlich sollen sämtliche Produkte direkt aus der Entwicklungsumgebung heraus nutzbar sein, so dass die Forderung nach Integrierbarkeit ein wesentlicher Punkt ist.

Diese Arbeit beschreibt das Vorgehen zur Ermittlung von Anforderungen an eine Lösung sowie eine prototypische Umsetzung im Java-Entwicklungsumfeld, wobei das Vorgehen auf andere Entwicklungsumfelder übertragbar ist.

Sascha Kluth

Title of the paper

Introduction of an Open Source Build Management and Continuous Integration system into a Java developer environment

Keywords

Build Management, Continuous Integration, Quality management, Version control, Test, integrated developer environment

Abstract

Having treated the basic concepts of Build Management and Continuous Integration products are researched by which the vision of an integrated Build Management and Continuous Integration system can be implemented. The research is restricted to (usually OpenSource-) products usable free of charge. As there is no product available supporting all aspects of Build Management and Continuous Integration, the research starts with the available Continuous Integration products. Products for the Build Management and the version control are also researched. Finally, all products should be directly accessible from the development environment, so the demand for integration is is of considerable importance.

This paper describes the approach to the identification of requirements for a solution, and a prototypical implementation for a Java development environment, while the approach is transferable to other development environments.

Darstellungskonventionen

In dieser Arbeit werden folgende Darstellungskonventionen verwendet:

Ein- und Ausgaben in der Konsole

```
haw@desktop:~$
```

Quelltexte und Konfigurationsdateien

```
/**
 * Quelltexte und Konfigurationsdateien
 */
public class HelloWorld
{
  public static void main( String[] args )
  {
    System.out.println( "Hello World!" );
  }
}
```

Pfadangaben und Dateinamen

`/home/pfad/datei.txt`

Die Beschreibung der „Prototypischen Umsetzung“ in Kapitel 6 und die Installationshinweise im Anhang unter 1 „Installationsbeschreibungen“ verwenden Platzhalter für Pfadangaben: `[server]` bzw. `[client]`. Diese sind jeweils, bei Nutzung von Windows durch „`D:\haw\server\`“ bzw. „`D:\haw\client\`“, respektive unter Linux durch „`/haw/server/`“ bzw. „`/haw/client/`“ zu ersetzen. Statt der unterschiedlichen Separatoren (Windows „\“ und Linux „/“) wird in dieser Arbeit einheitlich der Unix-konforme Separator „/“ verwendet, sofern sich die Angaben nicht auf für Windows typische Punkte beziehen.

Glossareinträge

Das Glosar befindet sich im Anhang 5, Seite 91ff.

Inhaltsverzeichnis

1 Einleitung

1.1 Motivation

In einem qualitätsgesichertem Entwicklungsprozess werden verschiedene Stufen durchlaufen. Jede Stufe besitzt ihre eigenen Ziele, die erfüllt sein müssen, um den Prozess in der folgenden Stufe fortsetzen zu können. Stellt sich das Ergebnis einer Stufe als fehlerbehaftet heraus, so ist in die entsprechende Stufe zurückzukehren. Bereits erstellte Arbeitsergebnisse der höheren Stufen werden dabei nicht verworfen, müssen aber gegebenenfalls modifiziert und erneut überprüft werden. Abbildung 1 zeigt einen vereinfacht dargestellten Ausschnitt aus dem Entwicklungsprozess.

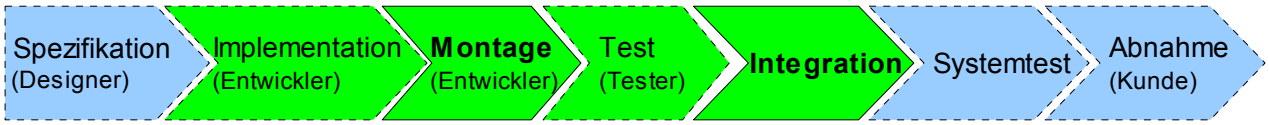

Abbildung 1: Vereinfachte Darstellung der Stufen eines qualitätssicherndem Entwicklungsprozesses

In der Spezifikationsstufe erstellen Softwaredesigner anhand von ermittelten Kundenanforderungen, technischen Rahmenbedingungen und weiteren Qualitätskriterien die Spezifikation der zu erstellenden Software. Die Umsetzung dieser Spezifikation erfolgt durch Entwickler in der Implementationsstufe, in der die Quelltexte für die Software erstellt werden. Die Quelltexte werden in der Montagestufe zu Komponenten montiert und in der Teststufe überprüft. Erfolgreich getestete Komponenten werden in der Integrationsstufe zu einem System zusammengefügt und in der folgenden Stufe einem Systemtest unterzogen. Abschließend erfolgt in der letzten dargestellten Stufe die Abnahme durch den Kunden.

Die Implementation wird im spezifischen Team mittels „Pair-Programming“ ausgeführt, bei dem immer zwei Entwickler nach dem Vier-Augen-Prinzip entwickeln und auch für die Erstellung und manuelle Ausführung der von ihnen unter Zurhilfenahme eines Testrahmens zu erstellenden Tests verantwortlich sind. Die Montage und Integration werden als lästige Routineaufgaben wahrgenommen und ebenfalls von den Entwicklern durchgeführt.

Diese Arbeit beschäftigt sich mit dem Bereich der Implementation bis zur Integration (grün) mit Fokus auf die Montage und Integration und erörtert hier auftretende Problemfelder und deren Lösungsmöglichkeiten.

Nach dem Leitbild „*Arbeitsplatz* für eigenverantwortliche Expertentätigkeit" (Züllighoven 2004) sind „Anwender eigenverantwortliche Experten, die qualifizierte Arbeit durch einen geeigneten *Arbeitsplatz* unterstützt erledigen“. In der Softwareentwicklung nimmt der *Entwickler* bei der Betrachtung seines *Arbeitsplatzes* diese Rolle des Anwenders, der eine *Entwicklungsumgebung* nutzt, ein. Züllighoven führt ebenfalls aus, das ein integrierter *Arbeitsplatz* eine bessere Unterstützung für den *Entwickler* bietet, als mehrere Teilarbeitsplätze. Daher soll die Integrierbarkeit der Lösung in die *Entwicklungsumgebung* der

Entwickler in dieser Arbeit als Grundvoraussetzung besondere Beachtung finden.

In dem Team, in dem der Autor im Rahmen der Erstellung dieser Arbeit mitgearbeitet hat, wird keine personelle Unterscheidung für die Übernahme der Rollen Entwickler und Tester getroffen, so dass die Teammitglieder die Anwender der in dieser Arbeit aufgezeigten Lösung sind und daher mit dem Begriff „Benutzer“ bezeichnet werden.

1.2 Andere Arbeiten zum Thema

Martin Fowler hat (Fowler 2002) hat die Idee der Contiunuous Integration formuliert und in einer Überarbeitung im Mai 2005 konkretisiert. Fowler beschreibt die Contiunuous Integration konzeptuell, ohne eine konkrete Lösung zu benennen.

Mike Clark erläutert (Clark 2004) die Grundzüge von Automatisierungen zur Prozessausführung. Dabei nutzt Clark die Kombination CVS als Versionskontrollsystem, ANT als Bulid Management System und CruiseControll für die Continuous Integration mit dem Fokus auf die Automatisierung. Fabian Dittberner (Dittberner 2006) baut hierauf mit einem konkreten Beispielprojekt auf und verwendet ebenfalls die Kombination CVS, ANT und CruiseControll.

Da sich die Marktsituation ständig verändert, insbesondere neue Produkte verfügbar werden und die oben genannten Arbeiten bereits vor einiger Zeit veröffentlicht wurden, scheint es sinnvoll, die Umsetzungsmöglichkeiten der Continuous Integration heute erneut zu überprüfen. Sowohl Clark als auch Dittberner gehen nicht auf die Integrierbarkeit der Build Management- und Continuous Integration Unterstützung in die *Entwicklungsumgebung* ein, was mit dieser Arbeit ergänzt werden soll.

1.3 Ziele dieser Arbeit

Ziel dieser Arbeit ist es, eine Vision zu entwickeln, wie eine vollständige Unterstützung für das Build Management und die Continuous Integration aufgebaut sein könnte, und diese Vision möglichst allen Anforderungen entsprechend umzusetzen.

Auch wenn die aufgezeigten grundsätzlichen Konzepte und Vorgehensweisen für unterschiedliche Vorgehensmodelle und Programmiersprachen nutzbar sind, werden sie in dieser Arbeit am Beispiel der *Entwicklungsumgebung* Eclipse für die Java-Entwicklung erläutert. Die Wahl einer weit verbreiteten Java-*Entwicklungsumgebung* bietet dabei für die Verdeutlichung der Konzepte und Vorgehensweisen die Vorteile, dass Java als Sprache zunächst plattformunabhängig ist. Somit müssen die Beschreibungen im Wesentlichen keine Rücksicht auf die verwendete Betriebssystemplattform nehmen und für eine Vielzahl von *Entwicklern* kann die vorgeschlagene Lösung direkt umgesetzt und genutzt werden.

Die Vorarbeiten zu dieser Arbeit haben ergeben, dass es zur Zeit noch kein frei verfügbares System für die *Entwicklungsumgebung* Eclipse gibt, das eine Build Management und Continuous Intergration Unterstützung bietet. Vielmehr steht eine Vielzahl von Produkten

zur Verfügung, die Teilaspekte des Build Managements und der Continuous Integration beschreiben. Diese Produkte auszuwählen und geeignet zu kombinieren ist nicht trivial und soll im Rahmen dieser Arbeit behandelt werden. Dabei geht es insbesondere darum, für die *Entwickler* einen *Arbeitsplatz* zu gestalten, an dem sie ihre Aufgaben besser unterstützt und Routinearbeiten möglichst automatisiert erledigen können, ohne für Teilaufgaben einen separaten *Arbeitsplatz* mit sich unterscheidenden Benutzungskonzepten nutzen zu müssen.

1.4 Umfeld

Diese Arbeit wurde im Rahmen einer Stipendiumstätigkeit bei der C1 WPS GmbH erstellt. Die C1 WPS ist ein mittelständisches Software-Architektur- und Technikbüro, das als universitäre Spin-Off-Firma des Arbeitsbereiches Softwaretechnik der Universität Hamburg 1999 gegründet wurde. Zur Zeit sind ca. 50 Mitarbeiter bei der C1 WPS beschäftigt und erstellen in Projektteams direkt beim Kunden Objekt Orientierte Software. Hierbei wird nach dem Leitbild WAM-Ansatzes (Züllighoven 2004) (siehe Kapitel 2.2 „WAM, Werkzeug-Automat-Material Ansatz“) vorgegangen, wobei das für Java verfügbare jWAM-Framework [JWAM] genutzt wird. Durch die weiterhin bestehende Nähe zum Arbeitsbereich Softwaretechnik sind die Mitarbeiter der C1 WPS in der Lage, neuste wissenschaftliche Erkenntnisse in ihre Projekttätigkeit einfließen zu lassen.

Im Rahmen ihrer Tätigkeit für die Deutschlandvertretung eines weltweit agierenden Autovermieters werden von Mitarbeitern der C1 WPS ca. ein Dutzend Anwendungsprojekte beim Kunden vor Ort entwickelt. Jedes dieser *Projekte* und deren Teilprojekte werden zur Zeit mit ANT-Scripten montiert und mittels JUnit-Tests [JUNIT] überprüft. Dabei kann der Montagevorgang einer umfangreicheren Anwendung bereits das Montieren vieler unterschiedlicher Teilprojekte mit komplexen Abhängigkeiten umfassen. Jedes *Projekt* kann sehr unterschiedliche *Montageziele* beinhalten. Je nach *Montageziel* sind die entstehenden Anwendungen unterschiedlich und zum Teil manuell zu testen.

Der WAM-Ansatz wirkt sich neben den bei der Beschreibung der Software benutzen Methaphern insbesondere im Bereich der Architektur auf die Qualitätssicherung und dadurch bei der Erhebung von *Softwaremaßen* aus. Auf die reine Ausführung des Montagevorganges hat er jedoch keinen Einfluss.

Bei der Nutzung des eingesetzten Versionskontrollsystems kommt es regelmäßig zur Abspaltung von Zweigen, die nach ihrer Weiterentwicklung in den Hauptzweig zurück integriert werden müssen. Die hierbei entstehenden Abhängigkeiten müssen beim Montieren und den *Tests* wieder Beachtung finden.

Mitarbeiter der EDV-Abteilung des Kunden haben jederzeit Zugriff auf sämtliche Quellen und Montageergebnisse, so dass auch bei in Bearbeitung befindlichen *Softwarebestandteilen* die Qualität möglichst hoch sein sollte, um das Vertrauen des Kunden in die entwickelte Software zu bestärken.

1.5 Argumentationsweg

In Kapitel 2 „Grundlagen“ werden Grundbegriffe erläutert und einige Konzepte zur Lösung von Qualitätssicherungsproblemen beschrieben.

In Kapitel 3 „Spezifikation“ werden die Anforderungen an eine *Entwickler* unterstützende Lösung zur *Integration* eines Build Management und Continuous Integration Systems ermittelt und gewichtet.

Daraus ergibt sich in Kapitel 4 die „Vision“ für eine Lösung.

Da noch kein einzelnes Produkt diese Unterstützung im Ganzen anbieten kann, wird in Kapitel 5 „Produktübersicht“ ein Überblick über verfügbare Produkte und deren Funktionsumfang gegeben. Hier wird auch eine Auswahl der Produkte auf Basis der in Kapitel 3 „Spezifikation“ ermittelten Anforderungen vorgenommen.

In Kapitel 6 „Prototypische Umsetzung“ werden die ausgewählten Produkte kombiniert. Es wird dargestellt, welchen Teilbereich der Vision sie ausfüllen.

In Kapitel 7 „Gegenüberstellung: Vision und Lösung“ wird das erreichte Ergebnis unter besonderer Berücksichtigung des Nutzens für den *Entwickler* analysiert .

Kapitel 8 „Fazit“ fasst die Ergebnisse der Arbeit zusammen und gibt einen Ausblick auf mögliche Weiterentwicklungen.

Genaue Konfigurationsanleitungen für die Prototypische Umsetzung aus Kapitel 6 befinden sich im Anhang 1 „Installationsbeschreibungen“.

2 Grundlagen

In diesem Kapitel werden die benötigten Begriffe und grundlegenden Konzepte dargestellt. Nach einer Einführung in die grundsätzlichen Abläufe der Softwareentwicklung und des WAM Ansatzes wird dargestellt, in welchen Szenarien *Entwickler* grundsätzlich arbeiten und welche Probleme sich hierbei ergeben können. Des Weiteren wird darauf eingegangen, wie die Automatisierung wiederkehrender Aufgaben, speziell Versionskontrolle, Test, Build Management und Continuous Integration die Probleme minimieren und so die Qualitätssicherung unterstützen können.

2.1 Ausschnitt aus dem Softwareentwicklungsprozess

Der allgemeine Arbeitsablauf für die Erstellung von Softwareprodukten stellt sich wie folgt dar:

Während der Planung wird in der Spezifikation unter anderem der architektonische Aufbau der zu erstellenden Software festgelegt, womit der grundlegende Aufbau der Software definiert ist. Dabei sollte auf bereits bekannte Architekturmuster (siehe (Gamma 1994) und (Fowler 2002)) zurückgegriffen werden.

Basierend auf der Spezifikation und Implementationsplanung (E1)[1] erstellen *Entwickler* die benötigten *Quelltexte* und *Tests* für einzelne *Softwarebestandteile* (E2-E18). Dabei ist zu beachten, dass die bereits beim Entwurf des *Softwarebestandteils* getroffenen Designentscheidungen adäquat umgesetzt werden. Neben den *Quelltexten* müssen auch die dazugehörigen *Tests* getestet (E15-E17) und auf Ihre Vollständigkeit überprüft werden. Diese Vollständigkeitsüberprüfung wird als *Testabdeckung* bezeichnet und kann zum Teil über *Softwaremaße* verglichen werden. Mit den Methoden der statischen Tests wird der *Quelltext* bereits vor der Ausführung auf Qualitätsmerkmale wie z.B. nicht erreichbaren Code oder zyklische Abhängigkeiten hin überprüft (E6).

Nach der Erstellung der *Quelltexte* werden sowohl die eigentlichen *Softwarebestandteile*, als auch die dazugehörigen *Tests* montiert und ausgeführt. Hierbei kann es zu *Fehlerwirkungen* kommen, die eine Modifikation der Quellen oder der *Tests* notwendig machen (E21). Zum Teil zeigt sich an dieser Stelle auch, dass die Spezifikation *Fehler* enthält, die dann zu beheben sind (E21). In diesem Fall beginnt der Prozess unter Beibehaltung der erstellten *Softwarebestandteile* erneut (E1). Sämtliche Bestandteile sind erneut auf ihre Spezifikationskonformität zu überprüfen.

Mit den Methoden der Fehlersuche und Behebung (engl. Debuging) werden Quellen und *Test* spezifikationskonform angepasst (E6 und E12). Lassen sich die *Softwarebestandteile* und *Tests* im geforderten Maße frei von *Fehlerwirkung* ausführen, so können die montierten *Softwarebestandteile* integriert werden (E29). Die Ergebnisse der dabei auszuführen-

1 Die in Klammern angegebenen Nummern beziehen sich auf die Übersicht in Anhang 3 „Ausschnitt aus dem Softwareentwicklungsprozess“.

den Integrationstests können wiederum eine Modifikation der bearbeiteten *Softwarebestandteile*, Integrationstests oder der Spezifikation notwendig machen. Ebenso kann es notwendig werden, auch die bei der *Integration* umgebenden *Softwarebestandteile* zu modifizieren (E33).

Sämtliche oben beschriebenen Schritte sind als potentiell optional zu betrachten. Welche Schritte ausgeführt werden, entscheidet der *Entwickler* an seinem *Expertenarbeitsplatz*.

2.2 WAM, Werkzeug-Automat-Material Ansatz

Der WAM-Ansatz (Züllighoven 2004) bietet eine Reihe von Metaphern für die Betrachtung von Softwareprodukten an.

2.2.1 Werkzeug

Ein *Werkzeug* ermöglicht es dem *Benutzer*, ein *Material* interaktiv zu bearbeiten. Dabei entscheidet allein der *Benutzer*, welche Bearbeitungsschritte in welcher Reihenfolge auszuführen sind. Für die Bereitstellung von Funktionalitäten kann ein *Werkzeug* auf die Nutzung von *Automaten* zurückgreifen.

2.2.2 Automat

Ein *Automat* bearbeitet *Materialien* aktiv nach einem zuvor exakt und vollständig vorgegebenen Plan ohne Benutzerinteraktion. Ein *Automat* kann allein durch zu übergebende Eingabedaten in seinem Verhalten modifiziert werden. Es ist möglich, dass ein *Automat* pausier- und fortsetzbar ist.

2.2.3 Material

Ein *Material* ist ein passives Arbeitsobjekt, das durch ein *Werkzeug* bearbeitet werden kann. Dabei wird ein *Material* (i.d.R.) durch sein Verhalten und nicht durch seine Struktur charakterisiert. Ein *Material* kann das Ergebnis der Arbeit sein oder durch Kombination mit anderen *Materialien* zum Arbeitsergebnis beitragen.

2.2.4 Behälter

Ein (fachlicher) *Behälter* kann *Materialien* aufnehmen, verwalten, ordnen und herausgeben. Ein *Behälter* fungiert also als nicht modifizierender Materialspeicher.

2.2.5 Arbeitsplatz

Ein *Arbeitsplatz* ist der Platz, an dem die zur Erledigung von Aufgaben benötigten *Werkzeuge* und *Materialien* zur Verfügung stehen. In dieser Arbeit wird von einem so genannten *Expertenarbeitsplatz* ausgegangen, einem *Arbeitsplatz* für *Benutzer*, die exakt wissen, welche Aufgaben sie auf welchem Wege erledigen wollen. Der *Expertenarbeitsplatz* unter-

stützt die vom *Benutzer* selbst gewählten Arbeitsabläufe bestmöglich, ohne Einschränkungen in der Bearbeitungsreihenfolge vorzugeben.

Abhängig davon, in welchem Szenario ein *Entwickler* arbeitet, muss sein *Arbeitsplatz* unterschiedliche Unterstützungen anbieten.

2.3 Szenarien

Für die Modifikation von *Softwarebestandteilen* sind unterschiedliche Szenarien vorstellbar, je nachdem, ob einzelne oder mehrere *Entwickler* in einem Team zusammenarbeiten und ob das Team in einem Raum oder getrennt arbeitet.

2.3.1 Einzelentwickler an einem Arbeitsplatz

Ein einzelner *Entwickler* modifiziert Bestandteile eines (Teil-)*Projektes*. Um zu verhindern, dass es zu Doppelbearbeitungen oder Bearbeitungsüberschneidungen kommt, muss der Einzelentwickler sich mit niemandem über den Zeitpunkt und die Reihenfolge von Modifikationen abstimmen.

Projektdateien werden ENTWEDER auf einem zentralem Server ODER im Dateisystem eines Rechners (oder eines sonstigen Datenträgers) ODER in einer Datenbank gespeichert. Auf jeden Fall erfolgt die Speicherung exklusiv an einem Ort.

Hierbei könnte bereits auf die Unterstützung eines Versionskontrollsystems zurückgegriffen werden, so dass dem *Entwickler* im zeitlichen Verlauf entstehende Zwischenversionen seiner Modifikationen zur Verfügung stehen.

2.3.2 Einzelentwickler an mehreren Arbeitsplätzen

Sobald ein *Entwickler* an mehreren Arbeitsplätzen arbeitet, kommt zu dem vorigen Szenario folgender Arbeitsschritt hinzu: Die zu modifizierenden Projektbestandteile werden in ihrer aktuellen Version an dem jeweiligen *Arbeitsplatz* lokal verfügbar gemacht. Dazu werden sie vom zentralem Speicherort kopiert. Die eigentliche Modifikation folgt dem obigen Szenario, bis schließlich nach Abschluss der Arbeit die Projektdateien wieder an dem zentralem Speicherort gesichert werden, aus dem sie stammen.

Nur durch das konsequente Einhalten der Bearbeitungssequenz (Datei laden, modifizieren, ZENTRAL speichern) für alle Projektbestandteile kann sichergestellt werden, dass keine konkurrierenden Modifikationen an unterschiedlichen Arbeitsplätzen implementiert werden. Hier bietet sich der Einsatz eines Versionskontrollsystems an. Der *Entwickler* arbeitet quasi in einem kooperativen Team, bei dem er in Personalunion sämtliche Teammitglieder darstellt. Dadurch ist der *Entwickler* weiterhin selbst dafür verantwortlich, dass es zu keinen konkurrierenden Modifikationen an den unterschiedlichen Arbeitsplätzen kommt.

2.3.3 Entwicklungsteam in räumlicher Nähe

Dieses Szenario folgt dem vorigen mit dem Unterschied, dass nun mehrere *Entwickler* gemeinsam an einem (Teil-)*Projekt* arbeiten. Die *Entwickler* müssen sich untereinander abstimmen, damit es bei den Modifikationen nicht zu Konflikten kommt.

In vielen Projektteams herrscht der Gedanke vor, dass bei einer übersichtlichen Anzahl von Projektbestandteilen und bei direkter Ansprechbarkeit der *Entwickler* die Abstimmung informell, durch kurzen Zuruf, geschehen kann. Da aber ohne konsequente Einhaltung eines formellen Verfahrens leicht *Fehler* passieren können, sollte die Entwicklung bereits in diesem Szenario durch ein automatisiertes Verfahren unterstützt werden.

Eine Möglichkeit hierfür ist die Verwendung von Sperren. Dabei sperrt ein *Entwickler* von ihm entnommene Projektbestandteile für den schreibenden Zugriff durch andere. So können andere *Entwickler* zwar mit dem bis dato aktuellen Stand weiterarbeiten, sie können ihn aber nicht ändern, bis der bearbeitende *Entwickler* seine Modifikation freigegeben und die Sperre entfernt hat. (Zu Sperren siehe folgendes Unterkapitel 2.4.)

Die Bearbeitung dieser Arbeit erfolgte in einem Team, dessen Arbeitssituation im wesentlichen diesem Szenario entspricht. Ein *Entwickler* hat jedoch über einen externen Zugang mitgearbeitet, so dass im Kapitel 6 „Prototypische Umsetzung“ das folgende Szenario Anwendung findet.

2.3.4 Verteilte Entwicklerteams

Sind die *Entwickler* räumlich getrennt, so muss für die Absprache ohne formelle Unterstützung noch ein Kommunikationsmedium genutzt werden, was zu Verzögerungen durch die nicht permanente Erreichbarkeit aller Projektmitglieder führt. Durch die nicht formelle Kommunikation erhöht sich außerdem die Fehlerwahrscheinlichkeit. So ist eine formelle Unterstützung in Form eines Versionskontrollsystems als zwingend zu erachten.

Insbesondere, wenn im Rahmen von globalen Projekten Arbeitsabschnitte von einem Team zum nächsten weitergegeben werden, ist eine direkte Absprache nicht mehr möglich, da nicht alle Beteiligten permanent erreichbar sind („Programing with the sun“).

2.4 Sperren

Wie in (Zukunft 2006) beschrieben, sind Sperren eine Lösung für Probleme beim Mehrbenutzerbetrieb. Eine Sperre ist dabei ein Recht, auf ein Objekt zuzugreifen. Bevor ein *Benutzer* (oder ein *Automat* oder *Werkzeug*) auf ein Objekt (bzw. *Material*) zugreifen kann, muss eine Sperre erworben werden. Ein Sperrenverwalter sorgt dabei dafür, dass Sperren nur dann vergeben werden, wenn sie verfügbar sind.

Oftmals wird unterschieden zwischen Sperren zum Lesen und Sperren zum Ändern. Lesende Sperren können für mehrere Zugriffe gleichzeitig vergeben werden. Sperren zum Ändern können nur exklusiv vergeben werden und dies auch nur, wenn keine lesende

Sperre für dieses Objekt vergeben wurde. Eine erhaltene Sperre ist nach Bearbeitung des Objektes unverzüglich zurückzugeben, damit andere *Benutzer* ebenfalls die Möglichkeit haben, das Objekt bei Bedarf zu modifizieren.

Der Vorgang von der Erwerbung bis zur Rückgabe einer Sperre als logische Arbeitseinheit wird auch als Transaktion bezeichnet. Durch einen Transaktionsmanager wird sichergestellt, dass die einzelnen Transaktionen die in Tabelle 1 dargestellten ACID-Eigenschaften garantieren.

Eigenschaft	Bedeutung
A: Atomar	Die Transaktion wird entweder ganz oder gar nicht ausgeführt.
C: Konsistent	Die Transaktion hinterlässt einen korrekten Zustand. Auch, wenn eine Transaktion fehlschlagen sollte, gelten nach ihrer Beendigung sämtliche Vor- und Nachbedingungen.
I: Isoliert	Transaktionen beeinflussen sich nicht gegenseitig.
D: Dauerhaft	Die Wirkung einer erfolgreich abgeschlossenen Transaktion bleibt dauerhaft erhalten.

Tabelle 1: ACID-Eigenschaften

2.5 Arbeitsbereiche

Unabhängig davon, in welchem der obigen Szenarien die Softwareentwicklung stattfindet, gibt es bei der Modifikation von Projektdaten unterschiedliche Abgrenzungen der Arbeitsbereiche:

2.5.1 Disjunkte Arbeitsbereiche

Eine Modifikation wirkt sich nur auf eine oder nur auf nicht voneinander abhängende Projektbestandteile aus. So kommt es nicht zu Konflikten durch die Modifikation.

2.5.2 Nahe liegende Arbeitsbereiche

Die Projektbestandteile sind zwar größtenteils klar trennbar, sind aber in Randbereichen abhängig voneinander. Durch die Nutzung von Sperrmechanismen ist es noch möglich, Konflikte zu vermeiden, da einzelne Projektbestandteile quasi disjunkt einem *Entwickler* zugeordnet sind.

2.5.3 Überschneidende Arbeitsbereiche

Projektbestandteile werden nicht exklusiv einem *Entwickler* zugeordnet. Bei der Bearbeitung werden Projektbestandteile in unterschiedlicher Weise modifiziert. Die Modifikationen von mehreren zeitgleich arbeitenden *Entwicklern* können sich dadurch gegenseitig aufheben oder widersprechen. Ein Konflikt ist entstanden, der aufgelöst werden muss. Dies kann geschehen, indem einer Modifikation Vorrang vor der anderen gegeben wird, oder indem durch Zusammenführung der beiden Versionen eine neue Version entsteht, die eine Teilmenge beider Modifikationen enthält.

2.5.4 Maßnahmen zur Verminderung von Konflikten in Arbeitsbereichen

Um in nicht disjunkten Arbeitsbereichen entstehende Modifikationskonflikte zu vermeiden, können zum Beispiel folgende Maßnahmen ergriffen werden:

2.5.4.1 Häufige Aktualisierung

Bei sehr kurzen Aktualisierungsabständen sinkt die Konfliktwahrscheinlichkeit dadurch, dass andere *Entwickler* in der Zwischenzeit weniger wahrscheinlich an dem Projektbestandteil gearbeitet haben. So erhält man zeitlich disjunkte Arbeitsbereiche.

2.5.4.2 Feingranulare Aufteilung

Durch die feingranulare Aufteilung werden kleine Projektbestandteile exklusiv einzelnen *Entwicklern* zugeordnet. Dadurch können im eigenen Arbeitsbereich zwar Konflikte vermieden werden, aber bei der *Integration* mit anderen Projektbestandteilen können dafür auf einer höheren Integrationsebene Konflikte entstehen. So wird die Konfliktlösung nur verzögert oder abgegeben.

2.5.4.3 Massiver Sperreinsatz

Durch diese Maßnahmen ist es möglich, auch in eigentlich überschneidenden Arbeitsbereichen einen quasi disjunkten Arbeitsbereich herzustellen. Allerdings ist der administrative Aufwand relativ hoch.

Durch den massiven Einsatz von Sperren wird es möglich, dass immer nur ein *Entwickler* schreibenden Zugriff auf einen Projektbestandteil bekommt. Die anderen *Entwickler* können in der Zeit bis zur Aufhebung der Sperre nur lesend auf den Projektbestandteil zugreifen, was zwar die Anzahl der Konflikte deutlich verringert, aber zu Verzögerungen führen kann.

2.6 Versionskontrolle

Mittels eines Versionskontrollsystems (engl. Version Control System, VCS) wird sichergestellt, dass Änderungen an den Projektdateien, insbesondere an *Quelltexten*, erkennbar, reproduzierbar und umkehrbar sind. Jede Änderung, die dem VCS bekannt gemacht wird, erhält automatisch einen Zeitstempel, eine Versionsmarkierung und einen anzugebenden Benutzerkommentar, der die Modifikationen zum letzten bekannten Stand beschreibt.

So ist es möglich, bei Fehlentwicklungen oder Anpassungen an Anforderungsänderungen auf einen alten, archivierten Zustand einer Projektdatei zurückzugreifen. Verwenden mehrere *Benutzer* zeitgleich Projektdateien aus einem gemeinsamen VCS, so können daraus entstehende Konflikte durch das VCS erkannt werden.

Unterstützt das VCS Sperren, so kann eine Projektdatei für den Schreibzugriff gesperrt werden. So ist sichergestellt, dass die Projektdatei nur von einem *Benutzer* modifiziert

werden kann und Modifikationskonflikte reduziert werden.

2.7 Softwaremaße (Metriken)

Die Verwendung des Begriffs (Software-)*Metrik* ist sehr gebräuchlich, wenn man von *Softwaremaßen* spricht. Eine *Metrik* ist jedoch in einem Raum definiert und analysiert Distanzen. Der korrekte Begriff beim Vermessen von Software ist also ein *Softwaremaß*, das wie folgt definiert ist ([IEEE1061]) :

„Ein Software(qualitäts)maß ist eine Funktion, die eine Softwareeinheit in einen Zahlenwert abbildet. Dieser berechnete Wert ist interpretierbar als der Erfüllungsgrad einer Qualitätseigenschaft der Softwareeinheit.“

Üblicherweise werden im Softwareerstellungsprozess *Softwaremaße* in der Testumgebung und dem Build Management erhoben.

2.8 Tests

Tests, die die montierten Projektelemente automatisch überprüfen, bieten als qualitätssichernde Maßnahme die Chance, die Konformität eines *Softwarebestandteils* zu seinem spezifizierten Verhalten zu überprüfen. Je nach Qualität der *Tests* können so von grobem Fehlverhalten des *Softwarebestandteils* bis zur exakten Überprüfung der Schnittstellenkonformität diverse *Fehlerwirkungen* ermittelt und durch deren Behebung die Menge der *Fehlerzustände* reduziert werden. Die Fehlerfreiheit kann so zwar nicht garantiert werden, aber die Wahrscheinlichkeit, einfache und auch sich maskierende *Fehlerzustände* frühzeitig im Entwicklungsprozess zu entdecken, steigt. Somit kann die Qualität der Software insgesamt gesteigert werden.

Ein weitverbreitetes Testframework für Javaprogramme steht mit dem JUnit-Framework zur Verfügung. Siehe [JUNIT] und (Link 2005). JUnit unterstützt den *Entwickler* durch die Bereitstellung eines Rahmens zur Erstellung von *Tests*, mit denen insbesondere die Testkriterien (Spillner 2005) abgeprüft oder das Vorgehen nach dem von WAM geforderten *Test-First-Ansatz* unterstüzt werden können.

Das Thema Testen ist sehr komplex und wird in dieser Arbeit nicht als solches behandelt. Eine ausführliche Einführung in das Gebiet der Softwaretests bietet (Spillner 2005). Siehe hierzu auch [GTB], [GITest], [ASQFTest] und [ISQI].

2.9 Build Management

Unter Build Management versteht man den standardisierten Ablauf, der alle Bestandteile einer Softwarekonfiguration erzeugt, die nicht direkt durch den *Entwickler* erstellt (oder durch das System bereitgestellt[2]) sind.

2 z.B. werden sog. DLLs vom Betriebssystem zur Verfügung gestellt.

Wie bei der Nutzung von Software allgemein gefordert, soll auch dieser Ablauf *korrekt* und *reproduzierbar* sein. Es soll außerdem *automatisch* und effizient, also mit *möglichst geringem Aufwand* ausgeführt werden.

Korrekt
Für jeden *Softwarebestandteil* ist sicherzustellen, dass die richtigen Eingabedaten (*Quelltext*, vormontierte *Softwarebestandteile* etc.) verwendet werden und dass jeder Bestandteil eingebunden wird. Neben der Tatsache, dass einzelne Bestandteile technisch korrekt (denke insbesondere an *Compilerdirektiven*) montiert werden, ist hier im Besonderen der Punkt zu beachten, dass einzelne Bestandteile wiederum aus zuvor montierten Bestandteilen zusammen gesetzt sein können. Somit ist wichtig, dass für eine korrekte Montage des Gesamtprojektes auch die einzelnen Bestandteile in der richtigen Reihenfolge montiert werden.

Reproduzierbar
Für die gleichen Eingabedaten soll immer das gleiche Programm als Ausgabe erzeugt werden. Auch wenn an den Quellen in der Zwischenzeit weiter gearbeitet wurde, soll durch die Verwendung alter Quellen ein alter Programmstand montierbar sein.

Automatisch
Neben dem einfachen Erzeugen der Programmbestandteile und letztlich dem Programm sollen nach der anfänglichen Konfiguration eines *Projektes* vom *Benutzer* keine Eingaben mehr erforderlich sein. Das Build Management soll den *Benutzer* insbesondere davon entlasten, Programmvarianten (z.B. für unterschiedliche Zielplattformen, Kundengruppen etc.) manuell zusammenstellen zu müssen. Das Build Management stellt sicher, dass für die Zusammenstellung eines *Montageziels* die richtigen Bestandteile zusammengeführt werden. Automatisch bedeutet auch, dass das Build Management das Zusammenspiel von Codegeneratoren, Compilern, Linkern, Loadern und Kompressoren regelt.

Aufwandsminimierung
Das Build Management sorgt dafür, dass nur Arbeiten ausgeführt werden, die notwendig sind, ohne dabei die Punkte Korrektheit und Reproduzierbarkeit zu vernachlässigen. So können z.B. Bestandteile, deren *Quelltext* sich nicht geändert hat und die nicht von einem geänderten anderen Bestandteil abhängen, ohne Auswirkung auf das montierte Programm beibehalten werden. Insbesondere soll die Montage von Teilprojekten betrachtet werden, um Änderungen im aktuellen Bearbeitungskontext montieren zu können, ohne den für Teilprüfungen unbenötigten Teil des Programms (eventuell langwierig) zu montieren.

2.10 Buildenvironment

Unter Buildenvironment wird in dieser Arbeit der Teil des *Arbeitsplatzes* verstanden, in dem der *Benutzer* das Build Management nutzt. Heute sind Quelltexteditoren, Testumgebung und Buildenvironment als integrierte Produkte verfügbar.

2.10.1 Java Buildenvironment

Eine weit verbreitete *Entwicklungsumgebung* für die Erstellung von Java-*Quelltexten* ist das OpenSource-Produkt Eclipse [Eclipse], das neben der Integration des Build Managements auch die Unterstützung für eine lokale Versionskontrolle enthält. Da Eclipse über einen Erweiterungsmechanismus verfügt und durch sog. PlugIns beliebig erweitert werden kann, bietet Eclipse den Vorteil der Anpassbarkeit an die Benutzerbedürfnisse und das Potential zur Bereitstellung eines integrierten *Arbeitsplatzes*. Die Standardangebote in Eclipse sind zur Versionskontrolle ein integriertes CVS (s. Kapitel 5.3.2 „CVS“) und für das Build Management die Unterstützung zur Ausführung von ANT-Skripten (s. Kapitel 5.2.2 „ANT“). Es fehlt jedoch die direkte Unterstützung von externen Versionskontrollsystemen und die Erstellung von Build-Skripten. Eclipse bietet nur einen XML-formatierenden Editor, in dem man die Skripte selbst von Hand schreiben muss.

In Eclipse ist es also möglich, den Buildprozess direkt aus der *IDE* heraus zu starten, als Ergebnis das Programm ausführbar montiert zu erhalten und das Programm auszuführen.

2.10.2 Kundenspezifisches Java Buildenvironment

Bei der Autovermietung wird eine kommerzielle Distribution von Eclipse, myEclipse 3.2.1 [MYECLIPSE] genutzt. Die aktuell verfügbare Version 6.0.1 wird zur Zeit eingeführt. Im Rahmen dieser Arbeit wird auf die Version 3.2 der Eclipse Standardversion Bezug genommen, die mit myEclipse 3.2.1 und 6.0.1 mit den in dieser Arbeit relevanten Funktionalitäten identisch ist.

Die vorhandenen *Projekte* werden mit der Version 1.6.5 von ANT [ANT] montiert. Seit Dezember 2006 ist Version 1.7.0 verfügbar, das aber keine für die aktuellen *Projekte* signifikanten Verbesserungen aufweist. Es wurden bereits Erprobungen zur Umstellung des Build Managements von ANT auf Maven vorgenommen, aber wieder verworfen, da die von Maven benötigte Internetanbindung nicht immer gewährleistet war.

2.11 Integration

Unter (Software-)*Integration* versteht man das Einfügen eines *Softwarebestandteils* in eine Software durch Verknüpfung mit anderen *Softwarebestandteilen*. So entsteht aus einzelnen *Komponenten* ein Gesamtsystem.

Die Fertigstellung einzelner *Komponenten* kann, je nach Projektgröße, Tage oder auch Monate auseinander liegen. Es gilt also zu definieren, in welcher Reihenfolge *Komponenten* integriert werden, um die *Integration* effektiver und damit Aufwandsminimiert und kostengünstiger vornehmen zu können. Für die *Integration* können unterschiedliche Strategien (Spillner 2005) angewendet werden:

2.11.1 Big-Bang-Integration

Die einzelnen *Komponenten* werden zwar jeweils getestet, aber erst integriert, wenn alle *Komponenten* fertiggestellt sind. Nun erst werden die Integrationstests durchgeführt, wodurch *Fehler* erst sehr spät im Entwicklungsverlauf erkannt werden. Da alle *Fehler* auf ein Mal auftreten, ist die Lokalisierung und damit auch die Behebung der *Fehlerzustände* aufwändig, zeitintensiv und damit kostspielig. Insbesondere zum Ende der Entwicklungszeit besteht häufig ein hoher Termindruck, der die *Tester* und *Entwickler* bei der Behebung unter weiteren Druck setzt. Vorteil dieser Strategie ist, dass weder Platzhalter noch Testtreiber benötigt werden.

2.11.2 Top-down-Integration

Die *Integration* beginnt mit den in der Hierarchie obersten *Komponenten*. Diese *Komponenten* werden von keiner weiteren *Komponente* aufgerufen, sie nutzen aber andere *Komponenten*, um ihre Aufgaben zu erfüllen. Der Vorteil der Top-Down-Integration liegt darin, dass für die Integrationstests keine oder nur sehr einfache Testtreiber benötigt werden. Nachteilig ist jedoch, dass die noch nicht vorhandenen, in der Hierarchie tiefer liegenden, *Komponenten* durch Platzhalter ersetzt werden müssen, was teilweise sehr aufwändig sein kann.

2.11.3 Bottom-up-Integration

Die *Integration* beginnt mit den Basiskomponenten, die keine weiteren *Komponenten* aufrufen. So können die einzelnen *Komponenten* jeweils sofort getestet werden. Auch Zusammengesetzte oder von anderen *Komponenten* abhängige *Komponenten* werden ausschließlich aus bereits getesteten *Komponenten* zusammengesetzt, so dass sie auch sofort getestet werden können. Für den Aufruf der zu testenden *Komponenten* sind jedoch Testtreiber zu erstellen.

2.11.4 Continuous Integration

Wie Martin Fowler in [FOWLERCI] beschreibt, werden bei der Continuous Integration *Komponenten* sofort nach ihrer Fertigstellung integriert. Die *Integration* folgt dabei also einer „zufälligen“ Reihenfolge. Die „Zufälligkeit“ ergibt sich aus den nur zum Teil planbaren Fertigstellungszeitpunkten der *Komponenten*. Sobald eine *Komponente* ihren Komponententest bestanden hat, wird geprüft, ob sie zu einer bereits vorhandenen und getesteten *Komponente* passt, und der Integrationstest zwischen diesen beiden *Komponenten* ausgeführt. Wiederum mit dem so entstandenen Teilsystem integrierbare *Komponenten* werden mit diesem integriert, so dass nach und nach das Gesamtsystem erfolgreich getestet vorliegt.

Nachteilig an der Continuous Integration ist zwar, dass sowohl Platzhalter, als auch Testtreiber benötigt werden, aber dafür werden die *Komponenten* zum frühestmöglichen Zeit-

punkt in ihre passende Umgebung integriert und in ihr getestet. Die so sehr frühzeitig gefundenen *Fehlerzustände* können umgehend korrigiert werden, wodurch sich Folgefehler vermeiden lassen. Insgesamt ist der Aufwand hierdurch deutlich geringer, bei einem potentiell qualitativ besserem Endprodukt. Somit ist die Continuous Integration i.d.R. als die optimale Integrationsstrategie zu wählen.

2.11.5 Integration im Kundenprojekt

In dem Projekt, an dem der Autor bei seiner Arbeit bei der C1 WPS mitgewirkt hat, wurde eine Mischstrategie zwischen Big Bang Integration und Continuous Integration genutzt. Dabei wurden inhaltlich zusammenhängende Modifikationen lokal getestet und dann gemeinsam alle ½ bis 3 Tage integriert. Je nach Umfang der Modifikation ähnelte die gewählte Strategie mal mehr der Big Bang Integration und mal mehr der Continuous Integration. Da an der Bearbeitung lediglich zwei Gruppen von *Entwicklern* beteiligt waren, kam es nur selten zu Integrationskonflikten.

2.12 Abgrenzung Build Management, Continuous Integration

Im Gegensatz zum Build Management, das die Erstellung der *Komponenten* regelt, betrifft die *Integration* das Zusammenfügen der *Komponenten* zu einem Gesamtsystem. Dieser Umstand soll deswegen nochmals hervorgehoben werden, da Programme zum Build Management häufig Funktionen zur *Integration* anbieten und Programme für die Continuous Integration zum Teil Build Management Funktionen enthalten.

2.13 Automatisierung

Wie Mike Clark in der Einleitung zu (Clark 2004) hervorhebt, „sind automatisierte Vorgänge exakt, konsistent und wiederholbar. Menschen arbeiten nicht so akkurat wie Maschinen, wenn sie sich wiederholende Arbeitsabläufe tätigen. Während Maschinen solche Arbeitsabläufe vielfach wiederholend ausführen, ohne die geringste Modifikation am Ablauf vorzunehmen, neigen Menschen dazu, gelegentlich dieselben Dinge unterschiedlich zu erledigen.“ Dies erschwert die Fehlersuche, insbesondere, wenn die Unterscheidungen der Abarbeitungsvorgänge nicht exakt protokolliert werden. Weiter schreibt Clark, dass „Automatisierung die Notwendigkeit zur Dokumentation reduziert. Ein automatisch ablaufendes Skript enthält als solches seine Beschreibung. Wenn neue Mitglieder in ein Entwicklerteam aufgenommen werden, benötigen sie zur Ausführung des Arbeitsvorganges nur noch das Wissen, wie das Skript auszuführen ist.“ Es wird also sowohl die Einarbeitungszeit in den Arbeitsablauf reduziert, als auch die Ausführungszeit, so dass diese Zeit für die eigentlichen Entwicklungsaufgaben genutzt werden kann.

Laut Clark sollten Automatisierungen immer dann vorgenommen werden, wenn man „müde wird, etwas manuell zu erledigen“ oder wenn absehbar ist, dass der zu automatisierende Arbeitsablauf mehr als zwei Mal ausgeführt werden wird. Dabei sollte eine Abwä-

gung geschehen, so dass nicht mehr Zeit für die Automatisierung verwendet wird, als durch diese gespart werden kann.

Versionskontrolle, JUnit-Tests, Build Management und Continuous Integration sind solche Automatisierungen, die vielfach auf die immer gleiche Weise ausgeführt werden sollen und somit durch ihre Automatisierung auch zur Sicherung der Qualität beitragen. Sie entsprechen damit der WAM-Metapher der *Automaten* nutzenden *Werkzeuge*.

Zusammenfassung

Je komplexer das Szenario wird, in dem *Entwickler* arbeiten, und je komplexer die *Projekte* werden, die sie bearbeiten, desto notwendiger wird die Unterstützung des *Arbeitsplatzes* zur Versionskontrolle erstellter *Softwarebestandteile*.

Nach der spezifikationskonformen Erstellung der *Quelltexte* müssen diese analysiert, getestet und montiert werden. Nach erfolgreicher Testung der Montageergebnisse werden diese zu einem Gesamtsystem integriert. Die *Integration* sollte nach dem Konzept der Continuous Integration erfolgen.

Die Delegation von wiederkehrenden Arbeitsabläufen an *Automaten* gibt dem *Entwickler* die Möglichkeit, seine Kreativität und intellektuelle Schaffenskraft für die Entwicklung zu nutzen, statt sie mit Routineaufgaben zu verschwenden. Je besser ein *Arbeitsplatz* den *Entwickler* mit *Werkzeugen* unterstützt, desto besser kann er sich auf seine Kernaufgaben konzentrieren.

3 Spezifikation

Im vorangegangenen Kapitel wurde eine Auswahl von Problemfeldern im Softwareentwicklungsprozess dargestellt und aufgezeigt, welche qualitätssichernde Unterstützungsmöglichkeiten es prinzipiell gibt. In diesem Kapitel sollen nun die Anforderungen ermittelt und gewichtet werden, die an Softwareprodukte gestellt werden, mit denen sich ein *Arbeitsplatz* aufbauen lässt, der einen integrierten Arbeitsablauf unterstützt. Gleichzeitig werden die konkreten Anforderungen an eine Lösung hergeleitet.

3.1 Anforderungsanalyse

Um eine dem Entwicklungsumfeld entsprechende Spezifikation für eine Lösung aufzustellen, ist der direkte Kontakt zu den Mitarbeitern, die mit der Softwareentwicklung im Projekt, mit der Projektleitung und mit der administrativen Verwaltung der *Entwicklungsumgebung* betraut sind, zu suchen. Im direkten Gespräch sind die Bedürfnisse der jeweiligen Benutzergruppen zu ermitteln.

Die Anforderungen an eine Lösung wurden mit Hilfe des „Requirements Engineering“ erhoben. Die Anwender, also in diesem Fall die C1 WPS-*Entwickler*, wurden zunächst nach ihrem Arbeitsalltag befragt. Hierbei ging es darum, sich ein Bild davon zu machen, wie die Arbeitsabläufe bei der Softwareerstellung, insbesondere im Bereich der Montage und *Integration*, aussehen, welche speziellen Probleme dabei auftreten können und welche Gedanken es zu Lösungsmöglichkeiten gibt. Die Anwender wurden dazu angehalten, sämtliche Ideen frei zu äußern. Die daraus entstandenen Anforderungen wurden bewertet und klassifiziert in Anforderungen, die unbedingt zu erfüllen sind (A), die erfüllt werden sollten (B) oder die einen Zusatznutzen darstellen (C). Diese Klassifizierung wird als ABC-Analyse bezeichnet. Deren konkrete Ergebnisse werden in Kapitel 3.1.4 „ABC-Analyse“ dargestellt.

3.1.1 Analyse- und Bewertungsgespräche im Rahmen eines Autor-Kritiker-Zyklus

In vier ca. einstündigen Gesprächen mit drei *Entwicklern* sind zu Projektbeginn erste Anforderungen ermittelt worden. Nach Ordnung der unterschiedlichen Informationen zu den Arbeitsabläufen ist folgendes Bild vermittelt worden, das in weiteren Gesprächen nochmals zwei *Entwicklern* zur Bewertung dargestellt wurde, um ein korrektes Verständnis sicher zu stellen.

3.1.1.1 Istzustand

Die Nummern in Abbildung 2 wurden korrespondierend zur in Abbildung 5 dargestellten Lösung gewählt.

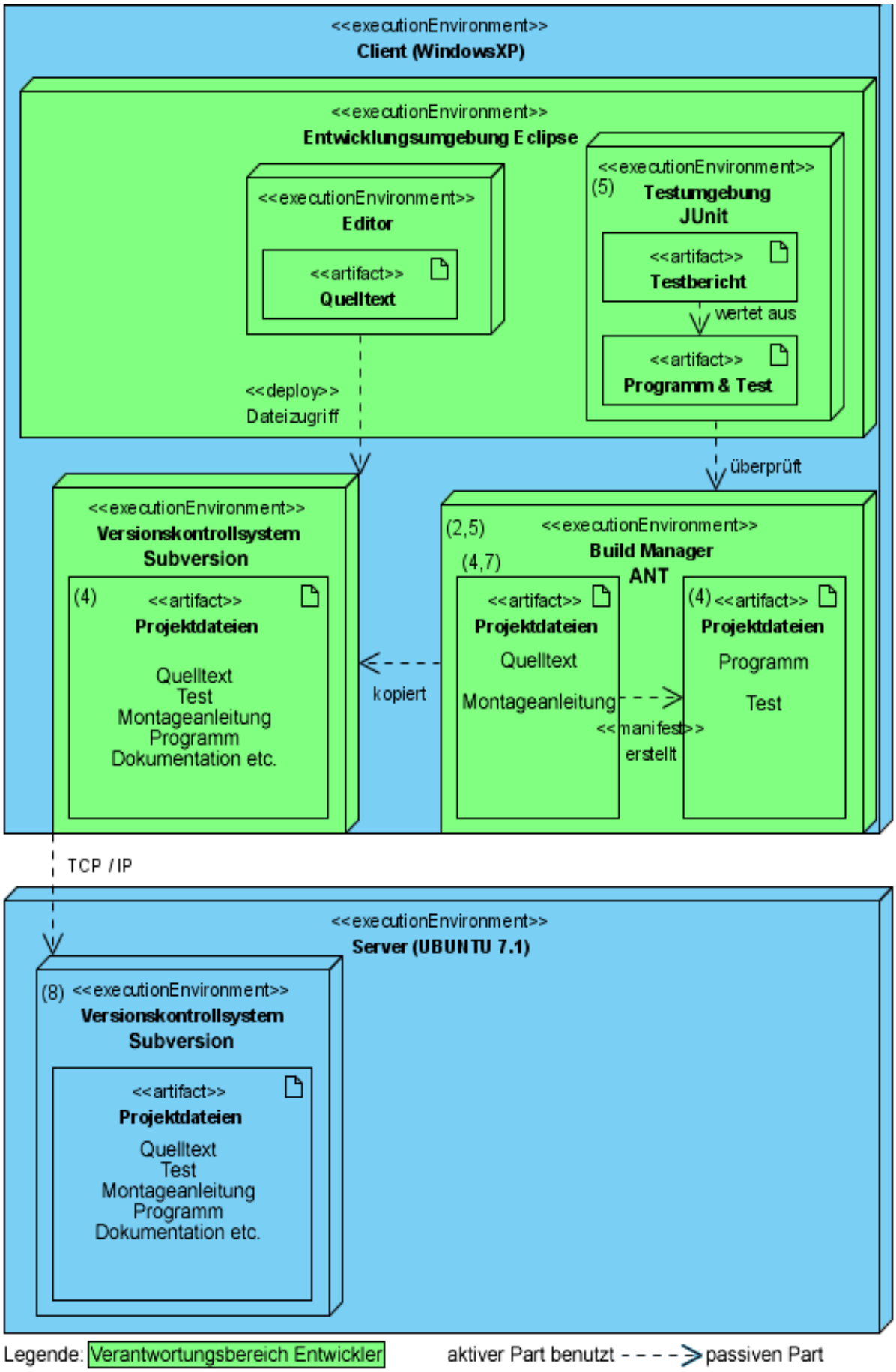

Abbildung 2: Schematische Darstellung des Istzustandes

Aktuell werden sämtliche Projektquellen in einem Subversion *Repository* gehalten. Die Projektbestandteile, die bearbeitet werden sollen, werden aus dem *Repository* kopiert und modifiziert. Falls nötig, werden die vorhandenen JUnit-Tests ergänzt, modifiziert, oder auch neu erstellt. Die geänderten Bestandteile werden mit Hilfe von ANT-Skripten montiert und die JUnit-Tests ausgeführt. Die ANT-Skripte werden im Prinzip in jedem neuen *Projekt* nahezu unverändert übernommen, so dass der Modifikationsaufwand hier gering ist.

Die JUnit-Tests sind von demjenigen *Entwickler* zu erstellen bzw. zu modifizieren, der die getesteten Programmbestandteile bearbeitet. Einen separaten Testbeauftragten gibt es nicht. Schlagen *Tests* fehl, werden die Problemstellen analysiert und je nach Fehlergrund die Programmbestandteile und/oder die *Tests* angepasst. Nachdem alle auf der *Entwicklungsumgebung* ausführbaren *Tests* erfolgreich abgeschlossen sind, werden die geänderten Projektbestandteile wieder in das *Repository* zurück kopiert. Das Testen, genauer das Warten auf den Abschluss der automatisch ausgeführten *Tests*, nimmt z.T. einen erheblichen Anteil der Entwicklungszeit ein. So werden zur Zeitersparnis i.d.R. nur die *Tests* ausgeführt, die die gerade direkt modifizierten Programmbestandteile testen.

Da auch Mitarbeiter von extern auf das *Repository* zugreifen und nicht immer die kopierten Bestandteile für Modifikationen anderer gesperrt werden, müssen von Zeit zu Zeit Konflikte bereinigt werden. Dies kann sehr aufwändig sein.

Es ist nicht immer sichergestellt, dass alle *Tests* erfolgreich durchgeführt worden sind, da auch fehlerhafte oder noch unvollständige Projektbestandteile in das *Repository* eingespielt werden können. Zum Teil werden durch Kommunikationsschwierigkeiten gegenläufige Entwicklungen vorgenommen. Dies geschieht jedoch nur in seltenen Fällen und nur in sehr geringem Maße.

3.1.1.2 Anregungen der Mitarbeiter für den Sollzustand

Die zur Zeit genutzten ANT-Skripte funktionieren zur allgemeinen Zufriedenheit und sollen nach Möglichkeit weiter genutzt werden, um den Migrationsaufwand gering zu halten. Die Implementation soll unabhängig von einer Onlineanbindung sein. Die *Entwickler* möchten nicht dazu gezwungen werden, jede Quelltextänderung in das *Repository* einspielen zu müssen, um sie testen zu lassen. Es soll möglich sein, einzelne *Tests* lokal zur Überprüfung von Modifikationen auszuführen. Es soll ebenso möglich sein, einzelne *Tests* vom Build Management und Continuous Integration System ausführen zu lassen, um z.B. zeitaufwändigere *Tests* an das Serversystem zu delegieren und die lokalen Ressourcen für Entwicklungsaufgaben zu nutzen. Dieselben *Tests* sollen sowohl lokal auf den Entwicklungsrechnern, als auch auf dem Integrationsserver ausgeführt werden können. Denkbar ist der Einsatz eines lokalen Windows-PCs als Server oder die Verwendung eines vorhandenen Linux-Servers. Nach Möglichkeit sollen beide Wege offen gehalten werden. Nach Möglichkeit sollten weitere qualitätssichernde Maßnahmen in den Softwareentwicklungsprozess integriert werden können.

3.1.2 Anforderungen

Neben den in Gesprächen ermittelten Anforderungen ergeben sich weitere Ideen aus der eigenen *Entwickler*- und Projektleitererfahrung und den Dokumentionen der in Kapitel 5 „Produktübersicht“ betrachteten Produkte. Da diese Arbeit in einer Softwareentwicklungsfirma erstellt wurde, die sich im Besonderen um die Entwicklung benutzerfreundlicher Software und Möglichkeiten zur Qualitätssicherung gibt, ergeben sich ebenfalls weitere Ideen für Anforderungen.

So sollte es möglich sein, spezielle *Tests* in einem separaten Kontext auszuführen.

Datenbanktests sollten optional nur ausgeführt werden, wenn die Datenbank zum Testzeitpunkt die notwendigen Ressourcen zur Verfügung stellen kann. Werden bei Systemmigrationen große Datenbestände bewegt, so dass eine Testausführung zeitweise nicht möglich ist, oder das CI-System so stark belastet, dass die *Entwickler* auf die Integrationsergebnisse ihrer Arbeiten (auch aus anderen Projekten, die das selbe CI-System nutzen) zu lange warten müssten, so sollen einzelne *Tests* deaktivierbar sein. Bei der Migration vom Entwicklungssystem auf das Produktivsystem darf dieses nicht blockiert werden. Dabei ist zu beachten, das sowohl das Entwicklungssystem, als auch das Produktionssystem eventuell auf gemeinsame Ressourcen zurückgreifen können.

3.1.3 Technische Architektur

Zur späteren Lastverteilung wäre es vorteilhaft, wenn Teilkomponenten des Systems auf verschiedene Rechner verteilt werden könnten. Für die Lösung bietet sich daher eine Client-Server Lösung mit Proxyunterstützung an, bei der viele *Entwickler* durch einen oder mehrere Server bedient werden können.

3.1.4 ABC-Analyse

Bei der Betrachtung der Literatur sowie der von den *Benutzern* gewünschten und den aus dem Projektalltag erfahrungsgemäß wünschenswerten Funktionalitäten hat sich in Rücksprache mit den *Entwicklern* die im Anhang 2 „Ergebnisse der ABC-Analyse“ dargestellte Priorisierung ergeben. Die Priorisierung erfolgte dabei nach dem erwartetem Nutzen für die *Entwickler* ohne Schätzung des Realisierungsaufwandes. In der folgenden Tabelle 2: Komprimierte Ergebnisse der ABC-Analyse werden nur die A-Anforderungen wiedergegeben und kurz erläutert.

Nr	Beschreibung der A-Anforderungen
	Architektur
1	Das System sollte als Client/Server System nutzbar sein, um Montage und *Integration* zentral vornehmen zu können.
2	Die Montage sollte auch lokal auf dem Client ausführbar sein und dabei exakt die selben Montageergebnisse erstellen, wie der Server.
3	Die lokale Monatge soll auch ohne Server- und ohne Internetanbindung ausführbar sein.
	Proxyunterstützung
4	Da die zentrale Buildkomponente Bibliotheksabhängigkeiten selbstständig auflösen können sollte, wird eine Proxyunterstützung benötigt.
	Build Management
5	Die Montage soll automatisiert erfolgen.
6	Die Definition soll in der Syntax von ANT erfolgen.
7	Bibliotheksabhängigkeiten sollte das Build Management selbstständig auflösen.
	Testframework
8	Für Komponenten- und Integrationstests soll eine Unterstützung des JUnit-Frameworks gegeben sein.
	Benutzungsschnittstelle
9	Um das System im Betrieb warten zu können, muss es mindestens eine Benutzungsschnittstelle geben. Diese sollte direkt in die *Entwicklungsumgebung* integriert sein.
10	Um die Funktionen des Systems nutzen zu können, muss es mindestens eine Benutzungsschnittstelle geben. Diese sollte direkt in die *Entwicklungsumgebung* integriert sein.
	Projektmontage
11	Der Montageprozess kann aufwändig und langwierig sein und sollte daher unterbrechbar sein.
12	Um zu zeitnahen Ergebnissen zu kommen, sollte der Montagevorgang automatisch bei Änderungen der Quellen gestartet werden.
	Deployment
13	Das Montageergebniss soll später als Klassen im Dateisystem ausgeliefert werden.
	Fehlermeldungen
14	Fehlermeldungen zum Buildprozess müssen ausführlich und von hoher Qualität sein.
15	Fehlermeldungen des CI-Systems müssen sehr ausführlich und von hoher Qualität sein.
	Benutzerinformation
	Sowohl im Fehler- als auch im Erfolgs-Fall soll der Benutzer informiert werden.
16	Die Information soll als Mail versendbar sein.
17	Die Information soll direkt in der *Entwicklungsumgebung* angezeigt werden.
18	Es soll definierbar sein, welcher *Benutzer* welche Informationen erhält. Dazu soll auch angebbar sein, welche Details die Information enthält.
	Stabilität
19	Bei Konfigurationsfehlern soll der Montageprozess frühzeitig beendet/nicht gestartet werden.
20	Das System soll bei Fehlern weiter aus der *Entwicklungsumgebung* heraus erreichbar sein um Konfigurationsänderungen vornehmen zu können.
	Erweiterbarkeit
21	Um das System zukünftigen Anforderungen anpassen zu können, soll es erweiterbar sein.

Tabelle 2: Komprimierte Ergebnisse der ABC-Analyse

Zusammenfassung

Im Rahmen eines Autor-Kritiker-Zyklus werden die *Entwickler*, in ihrer Rolle als Anwender der *Entwicklungsumgebung*, nach Ihren Arbeitsabläufen und nach benötigten Unterstützungen befragt.

Die sich daraus ergebenden Anforderungen werden mittels einer ABC-Analyse gewichtet und durch die *Entwickler* bestätigt.

4 Vision

Nach Auswertung der ABC-Analyse und dem Gedanken der Continuous Integration folgend ist eine Lösung wünschenswert, in der

- in der *IDE* modifizierte *Softwarebestandteile* aus der *IDE* heraus lokal montiert (2,5)[3] werden können,
- die montierten *Softwarebestandteile* lokal getestet (5) werden können,
- die montierten *Softwarebestandteile* aus der *IDE* heraus in ein zentrales *Repository* (8) eingepflegt werden können (sobald der geforderte Qualitätsstand der *Softwarebestandteile* erreicht ist),

- durch eine automatisierte Anwendung (Server),
  - regelmäßig das zentrale *Repository* auf Modifikationen überprüft wird (10 – Build Manager)
  - im Bedarfsfall die modifizierten *Softwarebestandteile* zentral
    - montiert (10 – Build Manager),
    - getestet (10 – Testumgebung) und
    - integriert (Server) werden,
  - das Montage- und Testergebnis in der *IDE* dargestellt wird,
  - erfolgreich getestete Montageergebnisse sowie Test- und Montageberichte (auch bei Fehlschlägen) und automatisch erstellte Dokumentationen versioniert gespeichert werden (13).

So gewinnen die *Entwickler* Freiräume, die für die inhaltliche Arbeit genutzt werden können. Es wird sichergestellt, dass keine durch automatische *Tests* auffindbaren *Fehler* im zentralen *Repository* vorhanden sind. Die nächste Test- und Entwicklungsstufe kann so auf einem qualitativ höherwertigeren Teilprodukt aufsetzen.

3 Die angegebenen Zahlen beziehen sich auf die Abbildungen 3 und 4. Die Nummern wurden korrespondierend zur in Abbildung 6: Komponentendiagramm des erweiterten Beispielprojektes dargestellten Lösung gewählt.

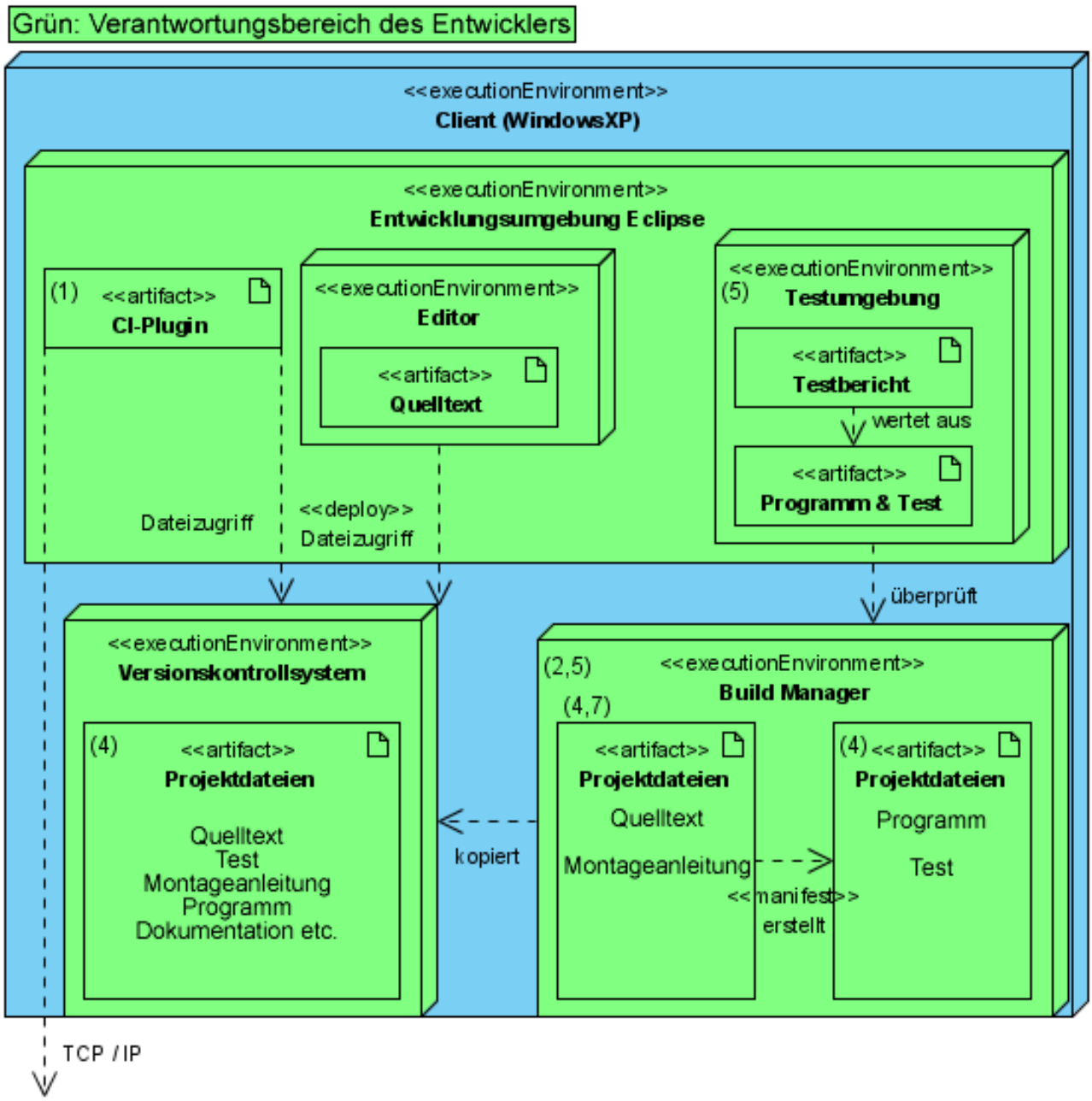

Abbildung 3: Schematische Darstellung der Vision – Client

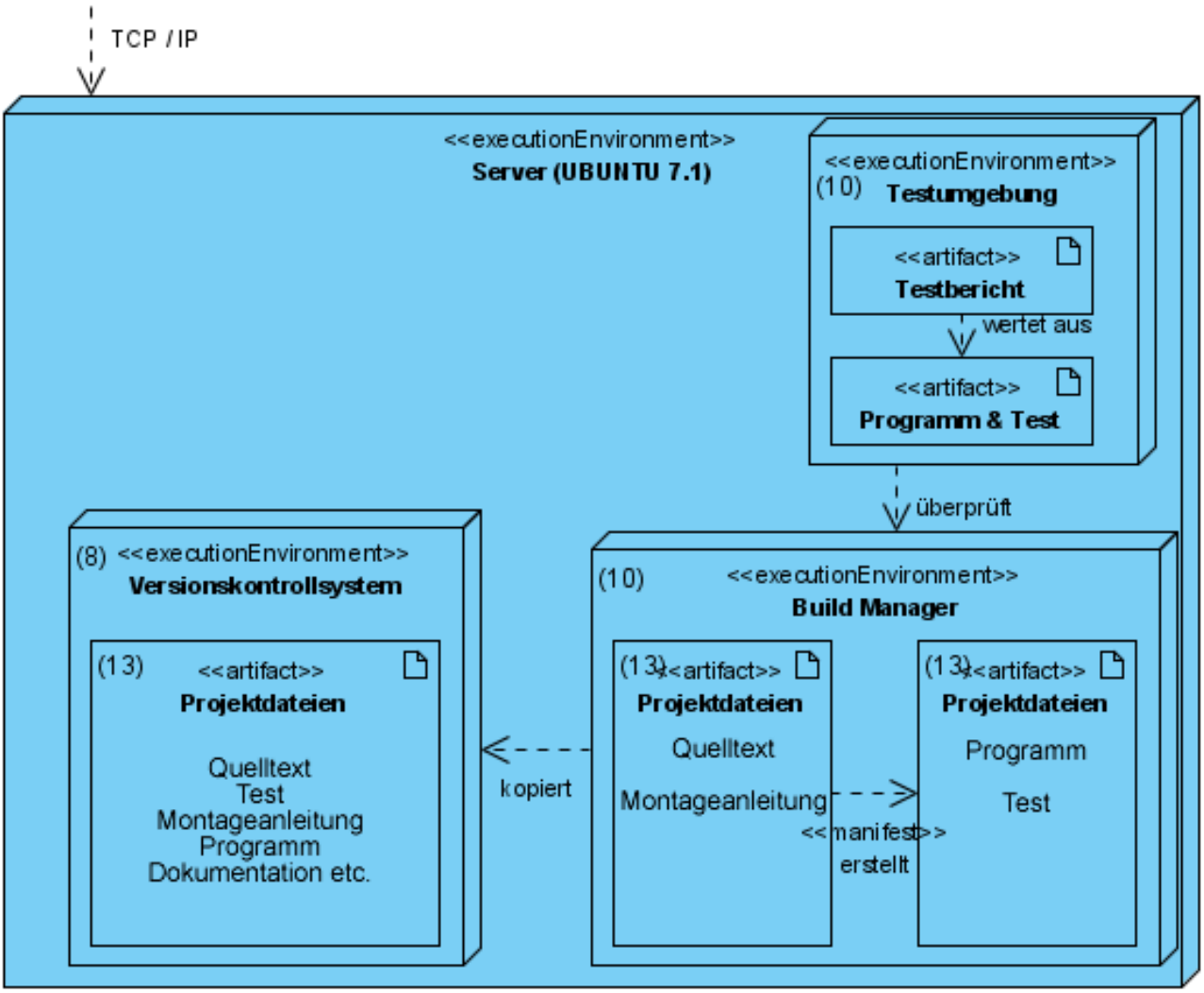

Abbildung 4: Schematische Darstellung der Vision – Server

Zusammenfassung

Auf Basis der im vorigen Kapitel aufgestellten Anforderungen wird eine Vision von einem integrierten und sowohl Build Management, als auch Contiuous Integration unterstützenden *Arbeitsplatz* entwickelt.

5 Produktübersicht

In diesem Kapitel werden die verfügbaren OpenSource-Produkte vorgestellt, die für die Realisation der in Kapitel 4 vorgestellten „Vision“ zur Verfügung stehen. Hier werden die Produkte detailliert vorgestellt, die auf Basis der in Kapitel 3 „Spezifikation“ aufgestellten Anforderungen ausgewählt wurden. Es wird dargelegt, warum die Verwendung alternativer Produkte verworfen wurde.

Allen in Frage kommenden Produkten ist gemein, dass sie zur freien Nutzung zur Verfügung stehen. Eine Übersicht über OpenSource Build Management & Continuous Integration Produkte für den Java-OpenSource-Bereich findet sich unter [OPENSOUCEBUILDSYSTEMS], die als Einstieg für die Recherche genutzt wurde.

5.1 Continuous Integration Server

Auf dem Markt gibt es diverse kommerzielle und OpenSource-Produkte, die Continuous Integration unterstützen. Einige wenige Produkte, wie z.B. AntHillOS [ANTHILL], oder Luntbuild [LUNTBUILD] werden sowohl in einer kommerziellen, als auch in einer funktionsreduzierten kostenfreien Version angeboten. Diese Arbeit konzentriert sich auf die kostenfrei verfügbaren Produkte mit Unterstützung für Java-*Projekte*, die wiederum in Java umgesetzt sind. Es ist zwar genauso gut möglich, ein Java-Programm mit einem in einer beliebigen anderen Programmiersprache erstellten Continuous Integration Server zu montieren und zu integrieren, es soll aber die Möglichkeit offen gehalten werden, das verwendete Betriebssystem frei zu wählen, ohne zusätzliche Abhängigkeiten z.B. von benötigten Bibliotheken zu erzeugen. Hier bietet sich Java dadurch an, dass es zum einen plattformunabhängig ist und zum anderen in der Javaentwicklung a priori Verwendung findet.

Gerade im Umfeld der professionellen Softwareentwicklung ist es wichtig, dass die eingesetzten Tools weiterentwickelt und gewartet werden. Dies ist bei OpenSource-Projekten mit einer aktiven Entwicklergemeinschaft eher wahrscheinlich als bei Projekten, die längere Zeit keine Weiterentwicklung erkennen ließen.

Unter [SERVERMATRIX] wurde mit Stand von Ende 2006 die damals aktuelle Marktsituation erhoben. Die dort angegebenen Fakten sind zwar zu überprüfen, bieten aber einen guten Überblick über potentiell verfügbare Produkte. Zum 28.7.07 und 21.12.07 hat eine eigene Recherche gegeben, dass keine weiteren Produkte zur Verfügung stehen. Auf dieser Auswahl basierend wurden als Java implementiert angegebene Produkte, in der jeweils aktuell verfügbaren Version, auf ihre freie Verfügbarkeit für den kommerziellen Einsatz überprüft. Die fünf Produkte sind:

5.1.1 Anthill

Anbieter: Urbancode Inc., http://www.anthillpro.com

Anthill benötigt (als einziges betrachtetes Produkt) zwingend einen Tomcat Server als Vorbedingung, um seine Ergebnisse über eine Webseite zur Verfügung stellen zu können und unterstützt als Build Management System ausschließlich ANT in den älteren Versionen 1.4 und 1.5. Der Einsatz der aktuellen ANT Version 1.7 ist ebenso wie die Wahl eines anderen Build Managers nicht möglich. Daher ist Anthill als veraltet anzusehen. Da der Anbieter keine Angaben über eine Weiterentwicklung von Anthill macht und eine kommerzielle Version AnthillPro anbietet, wird davon ausgegangen, dass Anthill nicht weiter entwickelt werden wird und vom Hersteller lediglich noch als Demoversion für AnthillPro dient. Somit scheidet Anthill für den Einsatz im Rahmen dieser Arbeit aus.

5.1.2 CruiseControl

Anbieter: Sourceforge, http://cruisecontrol.sourceforge.net

CruiseControl ist ein in Java implementiertes OpenSource-Projekt, das bei der Entwicklergemeinschaft sourceforge.org [SOURCEFORGE] angeboten wird. Als Grundlage seiner Interpretation von Continuous Integration stützt sich der Anbieter auf die Veröffentlichungen von Martin Fowler [FOWLERCI], womit eine konzeptuelle Nähe der in dieser Arbeit verwendeten Grundlagen gegeben ist. Seit der Veröffentlichung der Version 1 (29.3.01) bis zur Veröffentlichung der aktuellsten Version 2.7.1 (28.8.07) wurden 25 Zwischenversionen veröffentlicht (s. [CCCHANGELOG]), so dass dieses Projekt zur Zeit als aktiv gepflegt angesehen wird. Mit der CruiseControl Webseite, dem dort enthaltenden Wiki und angebotenen Mailinglisten steht eine gute und ausführliche Informationsbasis zur Verfügung. CruiseControl verfügt über einen eigenen PlugIn Mechanismus, über den insbesondere Qualitätssicherungserweiterungen integriert werden können.

5.1.3 Continuum

Apache Software Foundation, http://maven.apache.org/continuum

Continuum wird, ebenso wie die Build Management Systeme ANT, Maven und Maven2, von der Apache Foundation [APACHEFOUNDATION] angeboten, so dass Continuum bei der Kombination mit ANT oder Maven2 eventuell den Vorteil bietet, dass die Produkte besser zusammenarbeiten als die Kombination von Produkten unterschiedlicher Hersteller. Continuum wird mit integriertem Apache Webserver [APACHESERVER] angeboten. Man könnte vermuten, dass später einmal eine Continuum-Version mit ebenfalls integriertem ANT und Maven2 angeboten wird, so dass eine der Vision aus Kapitel 4 entsprechende integrierte Lösung zur Verfügung steht. Diesbezügliche Absichten der Apache Software Foundation sind aber noch nicht bekannt geworden.

Laut letztem Eintrag des Changelog zur aktuellsten Version 1.1. wurde die erste Änderung

Anfang 04 und die letzte Änderung Mitte November 07 vorgenommen, so dass das Produkt über einen längeren Zeitraum entwickelt wurde und die zum Download angebotene Version recht aktuell ist. Die Dokumentation der aktuellen Version 1.1 bietet (Stand 1.1.08) allerdings in weiten Bereichen noch als „to write“ markierte Leerstellen und die *FAQ* listet gerade einmal ein Dutzend Einträge auf, so dass die *Entwickler* zumindest mit der Bereitstellung von Informationen noch nicht das geforderte Maß an Unterstützung bieten. Gerade im professionellen Umfeld ist man beim Auftreten von Problemen auf eine schnelle Behebung angewiesen, um durch den Ausfall entstehende Kosten gering zu halten. Zur Zeit muss man bei Problemen in der Verwendung von Continuum auf weitere Ressourcen im Internet zurückgreifen, die auf der Anbieterseite auch nicht angegeben werden. Auch die Verwendung einer älteren Version hilft nicht weiter, da die Dokumentationen zu den alten Versionen weder in den zum Download angebotenen Archiven, noch auf der Webseite verfügbar sind.

Bei Erprobungsinstallationen kam es zu diversen Programmabstürzen, die im zeitlichen Rahmen von drei Manntagen nicht behebbar waren. Von einem produktiven Einsatz sollte daher zunächst abgesehen werden.

Continuum kommt auf Grund der mangelnden Dokumentation und der Installationsprobleme für den Einsatz im Rahmen dieser Arbeit nicht in Betracht.

5.1.4 Gump

Anbieter: Apache Software Foundation, http://gmp.apache.org

Gump wird ebenfalls von der Apache Foundation [APACHEFOUNDATION] angeboten und ist, entgegen der Angabe in [SERVERMATRIX], nicht in Java, sondern in Python [PHYTON] implementiert. Da Python für diverse Betriebssysteme zur Verfügung steht, schränkt das die Verwendbarkeit nur insofern ein, als dass neben Java noch eine weitere Laufzeitumgebung betreut werden muss. Gump ist auf eine separat anzubindende MySQL-Datenbank und unter Windows auf eine Posix-Umgebung (wie z.B. Cygwin [Cygwin]) angewiesen und steht nur als reine Sourcecodeversion zu Verfügung. Diese Punkte erhöhen den Installations- und Wartungsaufwand deutlich. Als Dokumentation steht ein Wiki auf der Gump-Webseite zur Verfügung, das auch aktiv erweitert wird. GUMP scheint z.Zt. noch ein Nischenprodukt für Python-Entwickler zu sein, dessen Funktionalitäten vom selben Anbieter mit Continuum angeboten werden.

5.1.5 Luntbuild

Anbieter: Luntbuild, http://luntbuild.javaforge.com

Luntbuild hebt sich durch die Möglichkeit der detaillierten Montageprozesssteuerung hervor, wobei Luntbuild nach folgendem Modell arbeitet: Projekte nutzen einen VCS-Adapter, der die Anbindung an eines von zwölf unterstützten VCS leisten. Builder definieren die Anbindung an eines von fünf Build Management Systemen. Schedules definieren Ausfüh-

rungszeitpunkte und die Abarbeitungsreihenfolge beliebig vieler Builder, deren Abhängigkeiten zu anderen Buildern und Projekten. Über den Umweg, dass man mittels Komandozeilenbuilder beliebige Programme ausführen kann, können auch differenzierte Backupstrategien etc. verwirklicht werden (siehe Anhang 1.5.12 „Kopie erfolgreich montierter Komponenten in Repository“). Luntbuild verfügt über eine eigene rudimentäre Benutzerverwaltung, Usermapping und die Möglichkeit, *Benutzer* über ein LDAP-kompatiblen Verzeichnisdienst zu authentifizieren.

Luntbuild bietet die Möglichkeit, die Benachrichtigungen seiner *Benutzer* detailliert über die Modifikation vorhandener Templatedateien zu gestalten, so dass Notifikationen den Bedürfnissen der *Benutzer* und der Corporate Identity angepasst werde können.

5.1.6 Vorauswahl Continuous Integration Server: CruiseControl und Luntbuild

Die fünf zunächst potentiell in Frage kommenden Continuous Integration Server wurden anhand der Herstellerdokumentationen und Testinstallationen auf Ihren Funktionsumfang hin verglichen. Die Vergleichsergebnisse werden in der folgenden Tabelle dargestellt, um die unterschiedliche Wahlfreiheit ergänzender Produkte des Build Management und der Versionskontrolle hervorzuheben und den zum Betrieb benötigten Vorbedingungen gegenüberzustellen.

Legende:
✓ Funktionalität vorhanden/Unterstützung gegeben
✗ Funktionalität nicht vorhanden/Keine Unterstützung
? Unbekannt
✓ Funktionalität vorhanden, aber nicht extern nutzbar.
⚠ Weitere Abhängigkeit über Java JDK, VCS und Build Management hinaus

Farbig hinterlegt, zur weiteren Betrachtung ausgewählte Continuous Integration Systeme

Funktionalität	Anthill	CruiseControl	Continuum	Gump	Luntbuild
Version Veröffentlicht	1.8.1 22.2.2006	2.7.1 4.9.2007	1.1 23.11.2007	? Version unbekannt, Revision 607873	1.5.3 19.10.2007
Implementiert	?	Java	Java	Python	Java
Integrierter BM	✗	✓ ANT 1.7	✗	✗	✗
Integriertes VCS	?	✗	✗	✗	✓ Subversion für interne Nutzung
Nutzbares externes VCS	✗	✓ Accurev ✓ Base Clearcase ✓ Clearcase UCM ✓ Cvs ✓ File system ✓ Perforce ✓ StarTeam ✓ Subversion ✓ Visual Sourcesafe	✓ Subversion	✓ Cvs ✓ Subversion	✓ Microsoft Visual Studio Team Foundation Server ✓ Perforce ✓ Plastic SCM ✓ SnapshotCM ✓ Star Team ✓ Subversion ✓ Surround SCM ✓ Visual Sourcesafe

Funktionalität	Anthill	CruiseControl	Continuum	Gump	Luntbuild
Integrierte DB	✗	?	✗	✗	✓ HSQLDB ✓ H2
Nutzbare externe DB	✗	✗	✓ MySQL	✓ MySQL	✓ Derby client/server ✓ HSQLDB ✓ H2 ✓ MySQL ✓ Oracle ✓ ProgreSql ✓ SqlServer
Nutzbarer BM	✓ ANT 1.4 ✓ ANT 1.5	✓ ANT 1.4 ✓ ANT 1.5 ✓ ANT 1.6 ✓ ANT 1.7 ✓ Maven ✓ Maven2 ✓ NANT ✓ Phing ✓ ShellScripte	✓ ANT 1.4 ✓ ANT 1.5 ✓ ANT 1.6 ✓ ANT 1.7 ✓ Maven ✓ Maven2	✓ ANT 1.4 ✓ ANT 1.5 ✓ ANT 1.6 ✓ ANT 1.7	✓ ANT 1.4 ✓ ANT 1.5 ✓ ANT 1.6 ✓ ANT 1.7 ✓ Maven ✓ Maven2 ✓ ShellScripte ✓ Rake
Integrierbar in Application-server	✓ Tomcat	✓ Jeder mit Servlet2.3 und JSP1.2 Unterstützung	✓ Tomcat ✓ JBoss ✓ Jetty ✓ Geronimo ✓ GlassFish	?	✓ Jeder mit Servlet2.3 und JSP1.2 Unterstützung
Integrierter Server	✗	✓ Jetty 5.1.3	? Apache vermutet	?	✓ Jetty >= 4.0
Verzeichnisdienstanbindung	✗	✓ LDAP über separaten Mapper	✗	✗	✓ LDAP Anbindung integriert
IDE Integration	✗	? Dashboard unterstützt ANT-Prozesse, CruiseControll nur in modifizierter Form	✗	✗	✓ Luntclipse-PlugIn
Tray Icon	✗	✓ JCCTray	✗	✗	✓ Aus Luntclipse-PlugIn
Clientprogramm	✗	✓	✗	✗	✗
Erweiterbarkeit	✗	✓ PlugIns	✓ Via Maven PlugIns	?	✓ Via Maven PlugIns
Weitere Abhängigkeiten neben Java JDK>=1.4	⚠ Tomcat			⚠ POSIX-Umgebung Unix,Linux oder Cygwin ⚠ "which" und "hostname" ⚠ Bash ab Version 3.0 ⚠ SSH-Client ⚠ Python ab Version2.4 ⚠ Subversion Client ⚠ MySQL ab Version 4 ⚠ MySQLdb & Pmock-Python Bibliothek	
Sonstiges				⚠ Blockiert bei Fehlern in angebundenem VCS oder Testumgebung	

Tabelle 3: Gegenüberstellung Funktionalitäten Continuous Integration Server

Continuum und Gump erfüllen noch nicht den für die Unterstützung professioneller Entwicklung notwendigen Reifegrad. Anthill ist wie oben dargestellt veraltet, so dass diese drei Produkte keine weitere Betrachtung finden.

CruiseConrtol und Luntbuild unterscheiden sich vom Funktionsumfang kaum. Luntbuild bietet gegenüber CruiseControl eine breitere Datenbankunterstützung und die enthaltene Möglichkeit zur Anbindung der Benutzerverwaltung an einen Verzeichnissdienst. Dafür verfügt CruiseControl über einen eigenen Erweiterungsmechanismus.

Die Anforderungen zur Integration der Benutzungsschnittstelle in die *Entwicklungsumgebung* (s. Tabelle 2: Komprimierte Ergebnisse der ABC-Analyse, Anforderung 9 und 10), erfüllt CruiseControl nicht, so dass Luntbuild für die Prototypische Umsetzung in Kapitel 6 gewählt wird. CruiseControl wird jedoch auf Grund der geringen Unterscheidungen zu Luntbuild weiter als Alternative betrachtet.

5.2 Build Management

Luntbuild und CruiseControl bieten kein integriertes Build Management an. Ein separates Build Management System muss über einen Adapter angebunden werden. Dafür stehen sowohl bei Luntbuild, als auch bei CruiseControl dieselben Versionen von ANT und Maven sowie Shell-Skripte zur Verfügung. NANT, Phing und Rake wurden zwar gesichtet, bieten aber keine für diese Arbeit relevanten Vorteile, so dass auf eine weitere Betrachtung zu Gunsten der leichteren Austauschbarkeit des Continuous Integration Servers verzichtet wurde.

5.2.1 Shell-Skripte

Shell-Skripte (unter Windows als Batch-Dateien bezeichnet) bieten die Möglichkeit, jedes über die Shell aufrufbare Programm auszuführen, so dass im Prinzip jedes beliebige Verhalten produzierbar ist. Mit Skripten, wie dem unter Unix weit verbreiteten „make“ lassen sich auch komplexe Montagevorgänge beschreiben. Sie sind aber in der Regel für das jeweilige Betriebssystem manuell anzupassen.

5.2.2 ANT

ANT bietet sich als vom Betriebssystem unabhängiger Skriptansatz an. ANT bietet die Möglichkeit, den Montagevorgang eines jeden Projektes detailliert zu beschreiben. Dies bietet zwar den Vorteil, dass auch Projektbesonderheiten im Montagevorgang direkte Beachtung finden können. Der *Entwickler* wird dadurch aber auch dazu gezwungen, sich mit dem gesamten Montagevorgang vertraut zu machen. Für den Einsatz beim Autovermieter stellt dies keinen Nachteil dar, da die Montageanleitungen in ANT-Skripten vorliegen und das Fachwissen vor Ort zur Verfügung steht.

5.2.3 Maven & Maven2

Maven wird, da sein Nachfolger Maven2 sich als verlässlich einsetzbar erwiesen hat, nicht weiter betrachtet.

Maven2 bietet durch einen deskriptiven Ansatz des Skriptaufbaus, eine Abstraktionsmöglichkeit gegenüber ANT-Skripten und unterstützt damit ebenfalls eine vom Betriebssystem unabhängige Lösung. Maven2 kann optional Abhängigkeiten verwendeter *Komponenten* und Bibliotheken selbständig auflösen und auf Wunsch nicht (in der benötigten Version) im Maven2 *Repository* befindliche Bibliotheken selbständig aus dem Internet nachladen. Maven2 bietet die Möglichkeit, *Softwaremaße* über integrierbare PlugIns während der Montage zu erheben, so dass hier ein Ansatzpunkt für eine weitere Qualiätssicherungsmaßnahme gegeben ist.

5.2.4 Auswahl Build Management: Maven2

ANT und Maven2 bieten beide die Möglichkeit, den gesamten Montageprozess zu begleiten. Während ANT sich um die Details kümmert, beschreibt Maven2 die Ziele abstrakt und bietet ein höheres Maß an Vererbungsmöglichkeiten und Vereinheitlichungen zwischen Projekten.

Sowohl ANT als auch Maven2 wird von allen oben vorgestellten Continuous Integration Servern unterstützt.

In der in Vorbereitung befindlichen Arbeit „Qualitätssicherung inkl. Programmierstilkontrolle mittels Maven2“ von Pergande (Pergande 2008) wird die Erweiterbarkeit von Maven2 und eine Auswahl von Maven2 PlugIns betrachtet, so dass an dieser Stelle nicht vertieft darauf eingegangen wird.

Die Integration von ANT ist in Eclipse bereits vorhanden, muss für Maven2 jedoch mittels eines PlugIns nachgerüstet werden.

In Abwägung der oben genannten Punkte wird Maven2 als Build Management System für die weitere Überprüfung gewählt.

Sollte sich die Umstellung der vorhandenen ANT-Skripte in Maven2-Skripte bei der Einführung in bereits vorhandenen Projekten als zu aufwändig erweisen, könnte für diese Projekte auf ANT zurückgegriffen werden, wobei die ANT-Skripte direkt aus Maven2 aus aufrufbar sind.

5.3 Versionskontrollsysteme

Luntbuild speichert die zu montierenden Projektdateien zwar in einem eigenen Subversion kompatiblen *Repository* ab, bietet aber keinen Serverdienst, dieses *Repository* als zentrales *Repository* zu verwenden. Deshalb wird (wie für CruiseControl auch) ein separates Versionskontrollsystem benötigt. Auch wenn auf Grund der Kompatibilität zu dem internen Versionskontrollsystem die Wahl von Subversion nahe liegt, werden noch weitere Alternativen betrachtet.

Unter [VERSIONSKONTROLLSYSTEME] findet sich eine Gegenüberstellung von ca. 20 Versionskontrollsystemen, es werden aber nur folgende von Luntbuild bzw. CruiseControl unterstützt:

Legende:

Fett: Sowohl von Luntbuild, als auch von CruiseControl unterstützt.
Fett und grau hinterlegt: Von Luntbuild und CruiseControl unterstützt UND frei nutzbar.

Unterstützt durch CruiseControl	Unterstützt durch Luntbuild	Anmerkungen
	Accurev	Kommerzielles Produkt
	Base Clearcase	Kommerzielles Produkt
	Clearcase UCM	Kommerzielles Produkt
	CVS	Open Source
	Dateisystem	
Microsoft Visual Studio Team Foundation Server		Kommerzielles Produkt
Perforce	**Perforce**	Frei nutzbar für 2 Nutzer und 5 Arbeitsplätze
Plastic SCM		Kommerzielles Produkt
SnapshotCM		Kommerzielles Produkt
Star Team	**StarTeam**	Kommerzielles Produkt
Subversion	**Subversion**	Open Source
Surround SCM		Kommerzielles Produkt
Visual Sourcesafe	**Visual Sourcesafe**	Kommerzielles Produkt

Tabelle 4: Vergleich von CruiseControl und Luntbuild unterstützter Versionskontrollsysteme

Die Schnittmenge der unterstützten Versionskontrollsysteme ist also Perforce, StarTeam, Subversion und Visual Sourcesafe, wobei Starteam und Visual Sourcesafe als kommerzielle Produkte ausscheiden. Perforce ist nur für eine sehr kleine Entwicklergruppe frei nutzbar, so dass Subversion als gemeinsam unterstütztes Versionskontrollsystem bleibt. Im Folgenden wird trotzdem kurz auf CVS als Alternative beim Einsatz von Luntbuild und VisualSVN Server als grafisch unterstützte Alternative für *Entwickler* unter Windows eingegangen.

5.3.1 Subversion

Subversion bietet auf einfache Weise eine Unterstützung für eine Versionskontrolle beliebiger Dateien. *Repositories* können über Ordnerstrukturen hierarchisch organisiert werden.

Subversion bietet sowohl die Möglichkeit, das *Repository* aufzulisten, als auch es zu durchsuchen. Es ist möglich, ältere Stände aus dem *Repository* zu laden. Die für die Nutzung von Subverison zur Verfügung stehenden Befehle sind für die Vielzahl von Betriebssystemen, für die Subversion zur Verfügung steht, nahezu identisch, so dass der Nutzer beim Systemwechsel in diesem Punkt keine Umstellungsschwierigkeiten erwarten muss, was wiederum die Flexibilität fördert und den Aufwand gering hält.

Der typische Workflow für die Nutzung eines Versionskontrollsystems wird von Subversion über Konsolenbefehle abgebildet:

Aktualisierung der eigenen Arbeitskopie

```
svn update
```

Änderungen einpflegen

```
svn add
svn delete
svn copy
svn move
```

Änderungen betrachten

```
svn status
svn diff
```

Änderungen rückgängig machen

```
svn revert
```

Konflikte bereinigen

```
svn update
svn resolved
```

Änderungen bestätigen

```
svn commit
```

Subversion sorgt beim Einsatz mehrerer Betriebsysteme für Dateiportabilität in der Art, dass die Dateien transportier- und speicherbar bleiben. In Unix-ähnlichen Systemen wird über ein Executionbit geregelt, ob eine Datei ausführbar ist. Diese Eigenschaft bleibt auch

beim Transfer über ein Windowssystem erhalten, das die Ausführbarkeit über den Dateinamen regelt und das Executionbit nicht unterstützt. Der originale Inhalt der Dateien bleibt erhalten. Automatische Umarbeitungen zum Beispiel von unterschiedlichen Zeilenvorschüben beim Kopieren zwischen Linux- und Windowssystemen nimmt Subversion nicht vor.

Subversion unterstützt Sperren von einzelnen Dateien und ganzen Ordnerhierarchien inkl. von Abhängigkeiten nach Zugriffsrechten. Der mitgelieferte Subversion Server lässt sich als Dienst installieren und unterstützt auch das immer weiter verbreitete WebDAV-Format [WEBDAV]. Subversion bietet die Möglichkeit, in einem *Repository* Zweige (engl.Branches) zu erzeugen und separat weiter entwickelte Zweige systemunterstützt wieder zusammenzuführen. Es können ebenso spezifische Änderungen unterschiedlicher Zweige zusammengeführt oder in einen neuen Zweig übernommen werden.

Subversion steht für eine Vielzahl von Unix, Linux, MacOS und Windowssystemen kostenfrei zur Verfügung. In Form eines OpenBooks (Collins-Sussman 2004) steht eine qualitativ hochwertige und ausführliche Dokumentation zur Verfügung. Ebenso stellt der Subversion Anbieter das Eclipse Integrations PlugIn SubClipse und die Windows Subversion *GUI* Tortoise zur Verfügung, so dass die Zusammenarbeit dieser drei Bestandteile gut funktionieren sollte. Zum gezielten Einsatz von Subversion siehe auch (Mason 2004).

Subversion bietet zwar über Konfigurationsdateien die Möglichkeiten zur detailierten Definition von Zugriffsrechten, eine Verzeichnisanbindung (z.B. an LDAP [LDAP] oder Active-Directory [AD]) steht nicht zur Verfügung.

5.3.2 CVS

CVS ist der Vorgänger von Subversion und bietet gegenüber Subversion einen deutlich kleineren Funktionsumfang. So bietet CVS zum Beispiel keine erweiterte Unterstützung für die Umbenennung und das Kopieren von Ordnern. Subversion bietet dies auch beim Transfer von einem Server zum anderen an. Für CVS ist die Unterstützung bereits in die Eclipse-*IDE* integriert, so dass kein zusätzliches PlugIn installiert werden muss. CVS lässt sich aber auch vom SubClipse PlugIn direkt nutzen. Da der Einsatz von SubClipse unproblematisch ist und einen deutlich höheren Benutzungskomfort bietet als die in Eclipse gegebene CVS Unterstüzung, ist die Kombination Subversion/SubClipse vorzuziehen.

5.3.3 VisualSVN Server

VisualSVN Server ist ein VCS, das laut Herstellerangaben den Anspruch erhebt, zu Subversion kompatibel zu sein. In den grundlegenden Funktionalitäten stimmt dies auch, aber es wird zum Beispiel das file:// Protokoll nicht unterstützt, so dass immer eine Serverlösung betrieben werden muss. VisualSVN Server ist ausschließlich als Windows-Version erhältlich und bietet keine Möglichkeit, Verwaltungsaufgaben der *Repositories* von der Konsole aus vorzunehmen, was die Integration mit anderen Produkten mangels weiterer Schnittstellen ausschließt. VisualSVN Server kann laut Herstellerangaben über jeden mit

Subversion kompatiblen Client bedient werden. In der Testinstalltion konnte dies für Tortoise (s. 5.4.1 Tortoise) mit dem vollen Subversion Funktionsumfang bestätigt werden. Außerdem ist VisualSVN Server in der Lage, bereits bestehende Subversion *Repositories* einwandfrei zu integrieren. So ist auch ein Parallelbetrieb von Subversion und VisualSVN Server möglich.

VisualSVN Server nutzt als Administrationswerkzeug die Microsoft Management Konsole und ist direkt in Visual Studio integrierbar, so dass es als Alternative zum Subverion Server beim Betrieb unter Windows eine gute Alternative darstellt.

5.3.4 Auswahl Versionskontrollsystem: Subversion

Da Subverison alle Anforderungen an ein Versionskontrollsystem, insbesondere die Anbindbarkeit an die Build Management Systeme ANT, Maven und Maven2, die Continuous Integration Server Luntbuild und CruiseControl und die Integration in die *Entwicklungsumgebung* erfüllt, sowohl über die Konsole als auch über einen eigenständigen Server ansprechbar ist und für diverse Betriebssysteme zur Verfügung steht, wurde Subversion für die prototypische Umsetzung ausgewählt.

5.4 Benutzungsschnittstelle für Versionskontrollsystem

Um Projektdateien, die nicht direkt in der *Entwicklungsumgebung* bearbeitet werden können (wie z.B. Textdokumente, Tabellen, Medien) im selben *Repository* verwalten zu können wie die unterstützten Projektdateien (wie z.B. *Quelltexte*, UML-Diagramme, erzeugte Programmdateien), bietet sich die Unterstützung des *Entwicklers* durch eine zusätzliche grafische Benutzungsschnittstelle für Subversion an. So müssen nicht alle Bearbeitungsschritte des *Repositories* von der Konsole aus ausgeführt werden.

5.4.1 Tortoise

Eine solche *GUI* ist Tortoise, ein *GUI*-Client für Windows, der sich bei der Installation direkt in den Dateimanager des Betriebssystems integriert. So ist es möglich, lokale Dateien ohne Versionskontrolle, lokale Dateien mit Versionskontrolle und entfernte Dateien mit Versionskontrolle an einem Ort auf gewohnte Weise zu verwalten. Für die Anbindung an externe Subversion-, CVS-, VisualSVN Server- und andere VCS-Server bietet Tortoise auch eine Proxyunterstützung. Da es sich bei der grafischen Benutzungsschnittstelle um keinen zentralen Bestandteil der Lösung handelt und Tortoise mit allen in dieser Arbeit evaluierten Versionskontrollsystemen zusammenarbeitet, wurden keine Alternativen für Tortoise evaluiert. Eine Alternative für Linux findet sich unter [KDESVN], eine Plattformunabhängige Java-*GUI* unter [JSVN].

5.5 Gegenüberstellung der Programme und Anforderungen

Abschließend werden die potentiell in Frage kommenden Continuous Integration Server und Build Management Systeme systematisch den Anforderungen aus Tabelle 2 gegenübergestellt. Die betrachteten Versionskontrollsysteme sind nicht nochmals aufgeführt, da sie keine A-Anforderungen betreffen.

Einige Anforderungen beziehen sich schwerpunktmäßig auf die Continuous Integration Server und andere eher auf Build Management Systeme, so dass die Einträge des jeweils anderen Systems entsprechend leer bleiben. Da CruiseControl durch die Nichterfüllung der Anforderungen zur Benutzungsschnittstelle (Anforderungen 9 und 10) ausscheidet, wurden die Funktionalitäten, die die Anforderungen 14,15,17-20 betreffen, nicht mehr evaluiert und mit ? gekennzeichnet. Somit ist Luntbuild als ausgewählter Continuous Integration Server bestätigt.

ANT und Maven2 unterscheiden sich im Rahmen der A-Anforderungen lediglich darin, dass Maven2 die Anforderungen 4 und 7 erfüllt und ANT im Gegenzug Anforderung 6 erfüllt. Nach erneuter Bewertung wurde die Anforderung 6 (ANT Syntax) als nicht so zu priorisieren eingestuft wie Anforderung 7 (Abhängigkeitsauflösung), so dass Maven2 für die „Prototypische Umsetzung“ gewählt wird. Da Maven2 aber auch in der Lage ist, ANT-Skripte mittels einer separaten ANT Installation auszuführen, könnte die Anforderung 6 im Nachgang noch erfüllt werden.

Somit steht als Auswahl für die „Prototypische Umsetzung“ die Kombination von Luntbuild, Maven2 und Subversion fest.

Legende:

✓ Funktionalität vorhanden/Unterstützung gegeben
✗ Funktionalität nicht vorhanden/Keine Unterstützung
? Unbekannt

Farbig hinterlegt, ausgewähltes Build Management und Continuous Integration Systeme

Nr	Beschreibung der A-Anforderungen	Luntbuild	CruiseControl	ANT	Maven2
	Architektur				
1	Das System sollte als Client/Server System nutzbar sein, um Montage und *Integration* zentral vornehmen zu können.	✓	✓		
2	Die Montage sollte auch lokal auf dem Client ausführbar sein. Möglichst in exakt der selben Art und Weise, wie bei der Servergestützten Montage.			✓	✓
3	Die lokale Montage soll ohne Server- und ohne Internetanbindung ausführbar sein.			✓	✓
	Proxyunterstützung				
4	Da die zentrale Buildkomponente Bibliotheksabhängigkeiten selbstständig auflösen können sollte, wird eine Proxyunterstützung benötigt.			✗	✓

Nr	Beschreibung der A-Anforderungen	Luntbuild	CruiseControl	ANT	Maven2
	Build Management				
5	Die Montage soll automatisiert erfolgen.			✓	✓
6	Die Definition soll in der Syntax von ANT erfolgen.			✓	✗
7	Bibliotheksabhängigkeiten sollte das Build Management selbstständig auflösen.			✗	✓
	Testsysteme				
8	Für die Komponenten- und Integrationstests soll eine Unterstützung des JUnit-Frameworks gegeben sein.			✓	✓
9	Um das System im Betrieb warten zu können, muss es mindestens eine Benutzungsschnittstelle geben. Diese sollte direkt in die *Entwicklungsumgebung* integriert sein.	✓	✗		
10	Um die Funktionen des Systems nutzen zu können, muss es mindestens eine Benutzungsschnittstelle geben. Diese soll direkt in die *Entwicklungsumgebung* integriert sein.	✓	✗	✓	✓
	Projektmontage				
11	Der Montageprozess kann aufwändig und langwierig sein und sollte daher an haltbar sein.	✓	✓	✓	✓
12	Um zu zeitnahen Ergebnissen zu kommen, sollte der Montagevorgang automatisch bei Änderungen der Quellen gestartet werden.	✓	✓		
	Deployment				
13	Das Montageergebnis soll später als Klassen im Dateisystem ausgeliefert werden.	✓	✓	✓	✓
	Fehlermeldungen				
14	Fehlermeldungen zum Buildprozess müssen ausführlich und von hoher Qualität sein.	✓	?	✓	✓
15	Fehlermeldungen des CI-Systems müssen sehr ausführlich und von hoher Qualität sein.	✓	?		
	Benutzerinformation				
	Sowohl im Fehler- als auch im Erfolgs-Fall soll der Benutzer informiert werden.				
16	Die Information soll als Mail versendbar sein.	✓	✓	✓	✓
17	Die Information soll direkt in der *Entwicklungsumgebung* angezeigt werden.	✓	?	✓	✓
18	Es soll definierbar sein welcher *Benutzer* welche Informationen erhält. Dazu soll auch angebbar sein, welche Details die Information enthält.	✓	?	✓	✓
	Stabilität				
19	Bei Konfigurationsfehlern soll der Montageprozess frühzeitig beendet/nicht gestartet werden.	✓	?	✓	✓
20	Das System soll bei *Fehlern* weiter aus der *Entwicklungsumgebung* heraus erreichbar sein, um Konfigurationsänderungen vornehmen zu können.	✓	?		
	Erweiterbarkeit				
21	Um das System zukünftigen Anforderungen anpassen zu können, soll es erweiterbar sein	✓	✓	✓	✓

Tabelle 5: Gegenüberstellung Funktionalitäten ausgewählter Continuous Integration und Build Management Systeme mit den A-Anforderungen der ABC-Analyse

Zusammenfassung

Ausgehend von der Feststellung, dass noch kein frei verfügbares Produkt angeboten wird, das die Vision des vorigen Kapitels umsetzt, werden verschiedene Programme bewertet, aus denen sich die Vision zusammensetzen ließe.

Dabei werden zunächst die frei verfügbaren Continuous Integration Systeme betrachtet. Aufgrund der großen Auswahl wird hier auf Basis eines funktionalen Vergleichs eine Einschränkung ohne exakten Abgleich mit den aufgestellten Anforderungen vorgenommen.

Es scheinen zunächst die zwei Continuous Integration Systeme CruiseControl und Luntbuild in Frage zu kommen, die mit diversen Build Management Systemen und Versionskontrollsystemen kombinierbar sind. So werden die jeweils von beiden Continuous Integration Systemen unterstützten Build Management Systeme betrachtet. Es stellt sich heraus, das Maven2 die Funktionalitäten von ANT und Shellskripten ebenso anbietet, darüber hinaus aber um weitere qualitätssichernde PlugIns erweiterbar ist und Bibliotheksabhängigkeiten selbstständig auflösen kann, so dass Maven2 der Kandidat für die prototypische Umsetzung ist.

Bei den Versionskontrollsystemen fällt die Wahl auf das einzige von beiden Continuous Integration Systemen unterstützte und frei verfügbare Subversion.

Die anschließende Gegenüberstellung der Anforderungen mit den noch in der Auswahl befindlichen Programmen zeigt als potentiell alle Anforderungen erfüllende Kombinationsmöglichkeit die Verbindung von Luntbuild, Maven2 und Subversion.

6 Prototypische Umsetzung

Wie im vorangehenden Kapitel dargestellt, gibt es noch keine umfassende, in einem Produkt vereinte Softwareunterstützung für die Continuous Integration. Anhand der in Kapitel 3 „Spezifikation“ aufgestellten Anforderungen und den in Kapitel 5 „Produktübersicht“ aufgeführten Funktionalitäten der verfügbaren Softwareprodukte wurden die geeignetsten Produkte für die Umsetzung der in Kapitel 4 vorgestellten Vision ausgewählt. In diesem Kapitel werden die gewählten Lösungsbestandteile kombiniert. Konkrete Installationshinweise befinden sich im Anhang unter 1 „Installationsbeschreibungen“.

6.1 Gewählte Produkte

Zur prototypischen Umsetzung wurden ausgewählt, für:

- die Versionskontrolle: Subversion
- die Netzwerkanbindung Versionskontrolle: Subversion Server
- den lokalen Zugriff auf die Versionskontrolle: Tortoise
- den Zugriff auf die Versionskontrolle aus der *IDE* heraus: SubClipse
- das Build Management: Maven2
- den Zugriff auf das Build Management aus der *IDE* heraus: m2eclipse
- die Continuous Integration: Luntbuild
- den Zugriff auf die Continuous Integration aus der *IDE* heraus: Luntclipse

Neben der eigentlichen *Entwicklungsumgebung*, in diesem Fall Eclipse, sind weitere acht Produkte und ein Browser notwendig, um dem *Entwickler* die gewünschte Unterstützung bis hin zur Continuous Integration anzubieten. Im Folgenden wird die Zusammenführung der einzelnen Produkte beschrieben. Dabei wird die Umsetzung in der Reihenfolge beschrieben, in der die Produkte installiert werden, da einzelne Produkte Basisdienste für andere Produkte zur Verfügung stellen. So werden die Produkte schrittweise ergänzt und die Unterstützung für den *Benutzer* wird immer weiter zur fertigen Lösung komplettiert. Dabei wird die Einführung unabhängig vom Entwicklungsbetrieb an einem Demoprojekt vorgenommen (s.a. Seite 53, 7.3 „Einführung zu Projektbeginn“). Es findet das unter 2.3.4 beschriebene Szenario „Verteilte Entwicklerteams“ Anwendung.

Abbildung 5 bietet einen Überblick über die Verteilung der einzelnen Programme auf Client und Server.

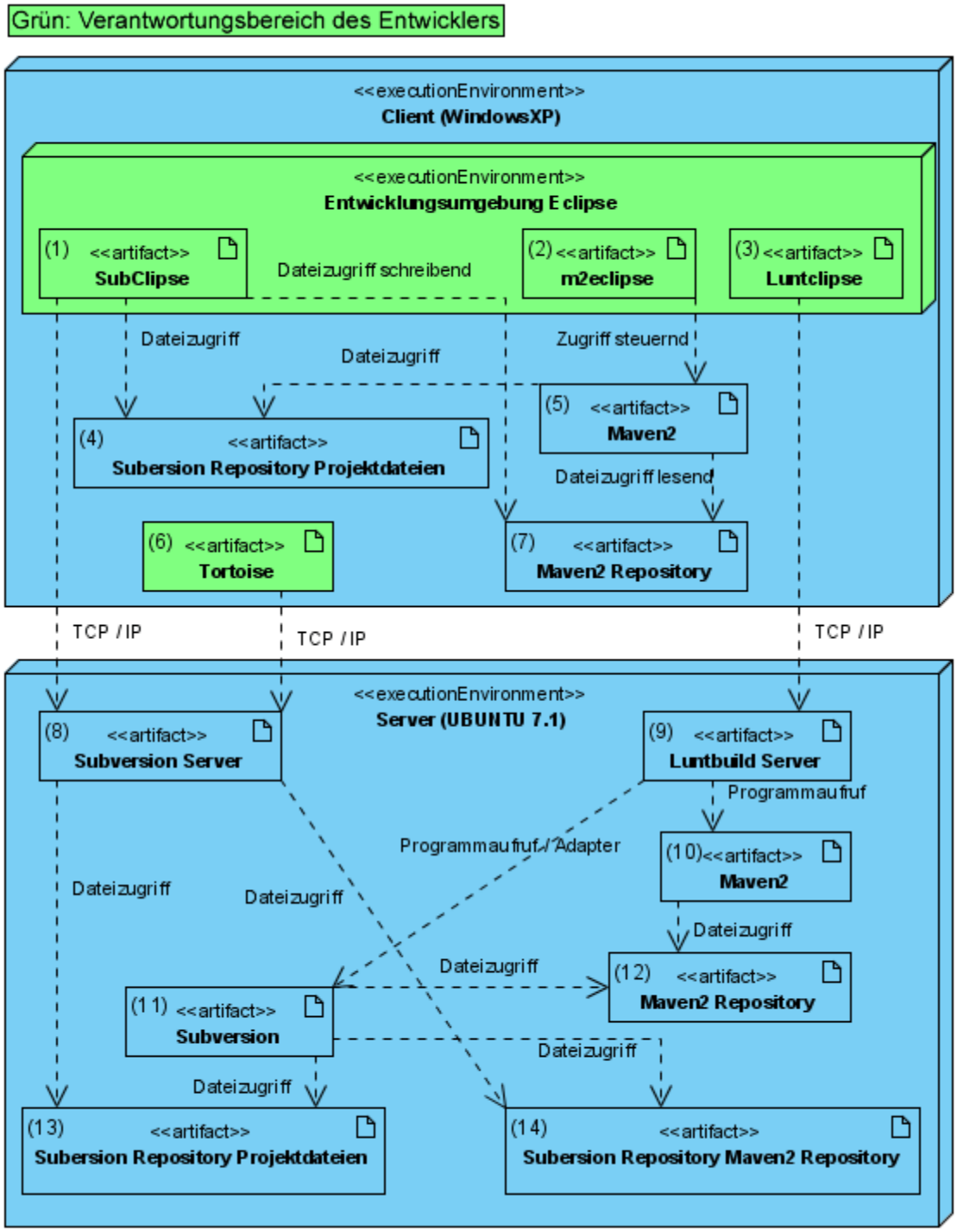

Abbildung 5: Verteilungsdiagramm der verwendeten Anwendungen

6.1.1 Aufbau der Versionskontrolle auf dem Server

Durch Subversion (11)[4] wird die Unterstützung der Versionskontrolle aufgebaut. Über die Konsolenbefehle von Subversion ist der gesamte Verwaltungs- und Nutzungsprozess durchführbar. Der Subversion Server (8) macht die Subversion *Repositories* (13 und 14) über das Netzwerk verfügbar. Nun ist es vom Client aus möglich, über Tortoise (6) ein erstes *Repository* anzulegen, das später das Demoprojekt aufnehmen soll.

6.1.2 IDE-Integration für Versionskontrolle – SubClipse

Eclipse bietet zwar die integrierte Nutzung von CVS *Repositories* an, aber es fehlt die Unterstützung von Subversion, so dass die Integration über ein PlugIn vorgenommen wird. SubClipse (1) ermöglicht die direkte Nutzung von Subversion *Repositories*, sowohl im lokalen Dateisystem (4) als auch über den Subversion Server (8). Dabei bietet die Integration neben einer eigenen Eclipse-Perspektive [Pers] auch die Unterstützung in den sonst bei der Entwicklung genutzten Perspektiven, wie z.B. der Hierachieansicht. Hier ist es möglich, über das Kontextmenü jede einzelne Datei oder ganze Ordnerstrukturen mit dem *Repository* abzugleichen. So kann nun auf das bereits erzeugte *Repository* (13) zugegriffen werden.

6.1.3 Aufbau des Build Management Systems auf dem Server

Durch die Installation und Konfiguration von Maven2 (10) entsteht nun das zentrale Build Management System. Maven2 baut ein eigenes *Repository* (12) auf, in dem es die zur Montage verfügbaren Bibliotheken bereithält. Dieses Maven2 *Repository* ist kein Subversion kompatibles *Repository*, sondern eine einfache Ordnerstruktur, in der die Bibliotheken abgelegt werden. Damit dieses Maven2 *Repository* in seinem jeweils aktuellen Stand dem *Entwickler* für lokale Montagen und dem Continous Integration Server (9) zur Verfügung steht, wird es mit Hilfe von Subversion (11) in ein (neu angelegtes) Subversion *Repository* (14) kopiert. So ist es nun möglich, zentral auf dem Server weitere Bibliotheken über Maven2 (10) in das Maven2 *Repository* (12) zu übernehmen und zu testen. Sind die Tests erfolgreich, so wird das Maven2 *Repository* (12) in das Subversion *Repository* (14) kopiert und dem *Entwickler* stehen die neuen Bibliotheken zur Verfügung.

Maven2 bietet die Möglichkeit, ein einfaches „Hello World“ Beispiel automatisch zu erzeugen. Dies wird genutzt, um die Installation zu überprüfen. Da das Beispielprojekt aber sehr trivial aufgebaut ist und keine Abhängigkeiten von *Komponenten* aufweist, wird es so erweitert, dass es aufrufende, aufgerufene und sowohl aufrufende als auch aufgerufene *Komponenten* enthält. So konnte belegt werden, dass Maven2 auch mit den unterschiedlichen Abhängigkeiten von *Komponenten* bei deren Modifikation umgehen kann (s. Abbildung 6).

4 Die in diesem Kapitel in Klammern stehenden Ziffern beziehen sich auf die Artefakte/Programme in Abbildung 5.

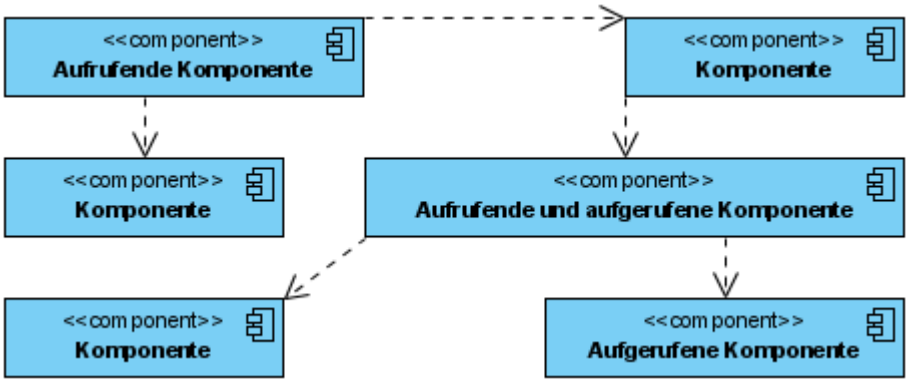

Abbildung 6: Komponentendiagramm des erweiterten Beispielprojektes

6.1.4 Aufbau des Build Management Systems auf dem Client

Auf dem Client wird Maven2 (5) ebenfalls installiert. Dies kann in diesem Fall durch ein einfaches Kopieren der Maven2 Serverinstallation (10) und Anpassung der Pfade geschehen. Das automatisch angelegte, lokale Maven2 *Repository* (7) wird durch das Maven2 *Repository* des Servers aus dem Subversion *Repository* (14) überschrieben. Damit die lokale Installation eigenständig arbeiten kann, wird sie in den OfflineMode gesetzt. Dadurch werden bei der Abhängigkeitsauflösung der Bibliotheken keine Bibliotheken aus dem Internet nachgeladen. Damit in *Projekten* neu benötigte Bibliotheken zur Verfügung stehen, sollen sie durch die Abhängigkeitsauflösung der Maven2 (10) Installation auf dem Server bereitgestellt werden. Neue Bibliotheken sind dann durch Subversion (11) auf dem Server vom Maven2 *Repository* (12) in das Subversion *Repository* (14) zu kopieren.

Dieses Kopieren ist zwar komplex, lässt sich aber vollständig automatisieren und bietet den großen Vorteil der zentralen Verwaltbarkeit der Bibliotheken. Dies ist wichtig, damit *Fehler* durch unterschiedliche Buildumgebungen auf dem Client und im Continuous Integration Server von vornherein vermieden werden können.

6.1.5 IDE-Integration für Maven2 – m2eclipse

Mittels m2eclipse (2) wird u.A. der Erstellungsdialog für neue *Projekte* um die Möglichkeit ergänzt, direkt neue Maven2 *Projekte* aus Eclipse heraus zu erstellen. Das Kontextmenü im Package Explorer wird erweitert, so dass sich für Maven2 *Projekte* die Abhängigkeitskontrolle verwendeter Bibliotheken und *Komponenten* direkt steuern lässt. Ebenso lassen sich die einzelnen *Montageziele* von Maven2 direkt aufrufen.

M2eclipse enthält zwar ebenfalls ein komplettes Maven2, das jedoch nicht genutzt wird, um die Maven2 Installation auf Client und Server einheitlich zu halten.

6.1.6 Aufbau des Continuous Integration Systems auf dem Server

Nach der Installation von Luntbuild wird der Luntbuild Server (9) gestartet. Nun steht über ein Webinterface eine grafische Benutzungsschnittstelle zur Verfügung, über die Luntbuild nun konfiguriert wird. Sämtliche folgenden Schritte können auch über Luntclipse (3) erfolgen. Es bietet sich jedoch auf Grund der größeren Darstellungsfläche des Webinterface an, dieses für die Erstkonfiguration zu nutzen. Bei eventuell auftretenden Konfigurationsproblemen muss so auch nicht herausgefunden werden, ob das Problem bei Eclipse, Luntclipse, oder Luntbuild selbst liegt.

Luntbuild baut seine Projekte so auf, dass ein Projekt zunächst einen VCS-Adapter erhält, über den das Versionskontrollsystem angebunden wird. In diesem Fall wird also ein Subversion-Adapter eingerichtet, der auf das in 6.1.1 erstellte *Repository* (13) zugreifen kann.

Nun benötigt ein Luntbuild-Projekt einen Build Management-Adapter, also einen Maven2-Adapter, der auf die Maven2 Installation (10) des Servers und die Montageanleitung des Demoprojektes aus dem Subversion *Repository* (13) zugreift. Von hier aus wird der Montageprozess gesteuert.

Luntbuild muss nun noch wissen, wann ein Projekt montiert und integriert werden soll. Dies wird über einen *Scheduler* ermöglicht, dem der Maven2 Adapter zugeordnet wird. Dabei gibt es viele Konfigurationsmöglichkeiten, wie zum Beispiel die Betrachtung von unterschiedlichen Projektabhängigkeiten oder Ausführungszeitpunkten. Es ist möglich, den *Scheduler* manuell zu aktivieren und dadurch einen manuell initiierten Montagevorgang zu starten. Um wirklich kontinuierlich zu integrieren, wird der *Scheduler* so konfiguriert, dass alle paar Minuten im Subversion *Repository* geprüft wird, ob Projektdateien geändert wurden. Falls dem so ist, soll der *Scheduler* den Montagevorgang starten.

6.1.7 IDE-Integration für Luntbuild – Luntclipse

Luntclipse (3) erweitert Eclipse um eine Luntbuild Ansicht, aus der heraus sämtliche Funktionen des Luntbuild Server (9) gesteuert werden können. Dabei können beliebig viele Server in die Luntbuild Ansicht aufgenommen werden.

In der Luntbuild Ansicht werden insbesondere die Status der unterschiedlichen Projekte dargestellt. Es ist möglich, Projekte manuell montieren zu lassen und sich eine historische Übersicht der Montageergebnisse anzeigen zu lassen.

Über einen integrierten Browser kann auch direkt auf die Web Schnittstelle der Server zugegriffen werden, falls Funktionen, die nicht in Luntbuild angeboten werden, aufgerufen werden sollen. Zum Beispiel könnte dies die Anzeige von Protokollen der in Maven2 erhobenen *Metriken* sein.

Die Möglichkeit der direkten Ergebnisanzeige von m2eclipse beschränkt sich auf die Textausgabe, so dass die grafische Ausgabe (über eine von Maven2 erzeugte Webseite) über Luntclipse möglich wird.

7 Gegenüberstellung: Vision und Lösung

7.1 Erreichter Stand

Auch wenn die Lösung sich im Aufbau deutlich von der Vision aus Kapitel 4 unterscheidet, erfüllt sie inhaltlich alle in Kapitel 3 aufgestellten A-Anforderungen, sowie viele B- und C-Anforderungen (siehe Tabelle 6, Anhang 2, Seite 80) und bietet darüber hinaus noch weitere Vorteile. Im Vergleich der Abbildungen 2 bis 5 wird deutlich, dass sich in der Vision der Verantwortungsbereich des Entwicklers gegenüber dem Ist-Zustand deutlich erweitert. Durch den geschickten Aufbau der Lösung kann der Verantwortungsbereich jedoch trotz des größerem Funktionsumfanges gegenüber dem Ist-Zustand deutlich reduziert werden.

Durch die Wahl von Maven2 als Build Management besteht die Möglichkeit, das System um die Erhebung von *Softwaremaßen* zu erweitern (siehe (Pergande2008)) und über den integrierten Aufruf von ANT-Skripten kann auch der zunächst nicht erfüllten Anforderung 6 (ANT Syntax für Montage) nachgekommen werden.

Durch die geschickte Nutzung der Online- und Offline-Modi von Maven2 und die automatisierte Verteilung der Maven2-*Repositories* wird ein über die Versionskontrolle der *Quelltexte* hinausgehende Vereinheitlichung der Montageumgebung ermöglicht.

Luntbuild ist in der Lage, die *Entwickler* von Montageergebnissen und auch von fehlgeschlagenen Montageversuchen zu unterrichten. Dies kann nicht nur, wie gefordert über die *Entwicklungsumgebung*, sondern zusätzlich auch über ein Tray-Icon und via eMail erfolgen.

Durch die Anbindbarkeit von Luntbuild an einen LDAP-kompatiblen Verzeichnisdienst wird eine zentrale Benutzerverwaltung möglich. Dazu muss jedoch auch Subversion mit der Benutzerverwaltung gekoppelt werden, was sich durch einen an den Verzeichnisdienst angebundenen HTTP-Proxy erreichen lässt. Diese Anbindung konnte vom Autor bereits in einem anderen Projekt mit Hilfe eines SQUID3-Proxys [SQUID] umgesetzt werden. (Dazu muss der Authentifikationsmechanismus jedoch selbst erstellt werden.)

7.2 Rückmeldungen der Entwickler

Nach einer Einarbeitung zweier Mitarbeiter in die neue Werkzeugunterstützung zum Build Management und der Continuous Integration konnten diese in der erstellten Lösung die Erfüllung der Anforderungen der ABC-Analyse (Kapitel 3.1.4) bestätigen. Die Lösung ist zwar im Aufbau komplex, was aber den *Entwicklern* als Anwendern verbogen bleibt. Die Nutzung ist gut zu handhaben, da die einzelnen PlugIns sich nahtlos in die *Entwicklungsumgebung* integrieren und die gewohnten Arbeitsabläufe unterstützen. Insbesondere die Möglichkeit, einzelne Arbeitsschritte sowohl in einer eigenen Eclipse-Ansicht zu erledigen, als auch über das Kontextmenü in der Hirarchie-Ansicht zu erreichen, entspricht der Un-

terstützung nach der Metapher des *Expertenarbeitsplatzes*.

Ein über die Anforderungen hinausgehendes Leistungsmerkmal, die Möglichkeit, *Softwaremaße* im Montagevorgang automatisch erheben zu lassen, konnte durch die Wahl von Maven2 angeboten werden. Ob der Einsatz von Maven2 in den spezifischen *Projekten* von Seiten des Kunden genehmigt wird und die Lösung (in der gewünschten Form) den Entwicklungsprozess unterstützen kann, wird leider erst nach Abgabe dieser Arbeit evaluiert werden können.

Es stellt sich für die *Entwickler* die Frage, in welcher Form das System bzgl. des Build Management eingeführt wird. Da die Continuous Integration bereits voll nutzbar ist und beim Build Management noch Absprachen mit dem Kunden zu treffen sind, zeichnen sich drei Möglichkeiten ab, das System einzuführen.

7.3 Einführung zu Projektbeginn

Kann die Einführung eines Build Management und Continuous Integration Systems zum Projektbeginn oder in der Projektvorbereitung erfolgen, so sind keine zusätzlichen Randbedingungen zu beachten, sofern dieses Projekt nicht von anderen abhängt oder andere von ihm abhängen.

Wenn das *Projekt* von anderen *Projekten* abhängt, so sollten diese möglichst ebenfalls stets aktuell im Subversion *Repository* vorliegen, damit bei der Montage auf diese Daten zugegriffen werden kann.

Hängen andere Projekte vom betroffenem Projekt ab, so ist sicher zu stellen, dass die anderen Projekte auf das Subversion *Repository* zugreifen können oder der Continuous Integration Server an einer für das andere *Projekt* zugreifbaren Stelle ablegt.

7.4 Einführung in bestehende Projekte

Bei der Einführung in bestehende *Projekte* müssen die *Entwickler*, während sie sich eigentlich auf die Erstellung des *Projektes* konzentrieren, zusätzlich die neuen Möglichkeiten und die Bedienung des neuen Systems erlernen.

Insbesondere muss auch überprüft werden, ob und in welcher Form das bestehende Build Management mit dem im Continuous Integration eingebundenen Build Management kompatibel ist. So ist es zum Beispiel möglich, dass aktuell mittels ANT montiert wird und nun während des Projektes ein Umstieg auf Maven2 erfolgt. Hierzu sollten in der Umstellung von *Projekten* von ANT nach Maven erfahrene Mitarbeiter zur Verfügung stehen.

7.5 Schrittweise Einführung

Bei der schrittweisen Einführung werden die Vorteile der Continuous Integration genutzt, aber das alte Build Management beibehalten. Bei Luntbuild gibt es dazu die Alternative, ANT-Skripte oder auch Kommandozeilenbefehle direkt in einem Build-Adapter aufzurufen. Dasselbe kann auch durch einen Aufruf aus einem Maven2 Montagevorgang erreicht werden.

So könnte zunächst kontinuierlich integriert werden, ohne zum Beispiel die in Maven2 einbindbaren Qualitätssicherungs-PlugIns zu verwenden und ohne die abstrakte Beschreibbarkeit des Montagevorganges von Maven2 zu nutzen. Bei Bedarf oder bei verfügbarer Zeit kann dann die Umstellung der ANT-Skripte auf Maven2-Skripte erfolgen.

7.6 Einführung beim Kunden

Für ein Testprojekt wurde das System bereits beim Kunden installiert. Um den *Entwicklern* die vollen Unterstützungsmöglichkeiten der Lösung zu bieten und der Forderung des Kunden nach Verwendung von ANT-Skripten für den Montagevorgang nachzukommen, ist folgendes Vorgehen möglich:

Die Lösung wird wie bereits installiert mit Maven2-Skripten genutzt. Für den Kunden werden die Montagen zusätzlich auf Basis der bereits vorhandenen ANT-Skritpe ausgeführt und separat zur Verfügung gestellt. Je nach den Bedürfnissen der *Entwickler* werden die Maven2-Skripte um PlugIns erweitert, so dass de Qualitätssicherung unterstützt und die *Entwickler* von lästigen Routineaufgaben befreit werden können. Durch die Parallelnutzung von zwei Skripten für einen Montagevorgang muss sichergestellt werden, dass die Skripte stets äquivalente Montageergebnisse liefern.

8 Fazit

Auch wenn die Lösung komplexe Konfigurationsvorgänge notwendig macht, so bietet sie, sobald man sich eingearbeitet hat, enorme Vorteile. Den *Entwicklern* werden lästige Routineaufgaben abgenommen, *Softwaremaße* können erhoben und zur Qualitätssicherung genutzt werden, und nicht zuletzt werden durch die gewonnene Zeit Kosten eingespart.

Die Lösung geht in einigen Punkten über die Vision heraus, ohne auf die Erfüllung von Anforderungen zu verzichten, und bietet außerdem noch einige Ansatzpunkte für weitere Unterstützung der *Entwickler*. Für den *Administrator* der Build Management und Continuous Integration Systems wäre insbesondere bei der Umstellung von ANT- in Maven2- Montagebeschreibungen eine bessere Unterstützung durch deutlich umfangreichere Beispiele und *Werkzeuge* wünschenswert. Ergänzend wäre die Angabe von begleitender Literatur auf der Maven2 Webseite wünschenswert. Darüber hinaus sind noch folgende Punkte im Laufe der Bearbeitung erkannt worden:

8.1 Offene Punkte

- Im aktuellen *Projekt* wurde die in m2eclipse integrierte Maven2 Installation durch eine separate Installation ersetzt. Zu klären wäre, ob dieser Vorgang zwingend notwendig ist.
- Für eine, wie in 7.6 „Einführung beim Kunden“ beschriebene Parallelnutzung von ANT und Maven2 Skripten wäre ein automatischer Abgleich der Skripte wünschenswert.
- Luntbuild ist erst nach einiger Eingewöhnung schnell zu benutzen, da die grafische Benutzungsschnittstelle nicht konsequent den bekannten Benutzungsmodellen folgt. So werden neue VCS-Adapter und Builder oben rechts über ein „Neu“-Icon erstellt, während Schedules über ein „Schedule Build“-Icon erstellt werden. Erwartet worden wäre, dass alle Elemente links oben über ein einheitliches „Neu“-Icon erstellbar sind. Dies sollte überarbeitet werden. Für die Evaluierung, welche Defizite bei der Benutzerführung von Luntbuild genau vorhanden sind, bietet sich eine Überprüfung in einem Usability-Labor [USELAB] an.
- Interessant wäre auch der im Rahmen dieser Arbeit nicht mögliche Abgleich mit kommerziellen Produkten.
- Wie in 7.1 „Erreichter Stand“ bereits erwähnt, muss für eine einheitliche Autorisierung noch der Umweg über die Verwendung eines Proxyservers als Anbindungsschnittstelle für Subversion gegangen werden. Wünschenswert wäre, dass das VCS bereits von Haus aus die Nutzung von Verzeichnisdiensten böte. Für das Build Management scheint dies nicht notwendig, da es hierbei um die Montagebeschreibung und nicht den Zugriff auf Quelldateien geht.

- In dieser Arbeit wurde nicht betrachtet, wie mit der Verwaltung von *Fehlern* umgegangen wird. Interessant wäre es, das Fehlermanagement so anzubinden, dass es aus der *Entwicklungsumgebung* heraus, mit Bezug auf die durch die Continuous Integration erstellten Ergebnisse, nutzbar ist.
- Es steht (z.B. mit [OMONDO]) bereits eine Möglichkeit zur Verfügung, in UML spezifizierte *Komponenten* (zu mindestens vom Gerüst her) in Eclipse automatisiert erstellen zu lassen. Interessant wäre es, inwieweit dies mit den für die Continuous Integration verwendbaren Build Management Systemen kompatibel ist.
- Für *Projekte*, in denen die Integration in die *Entwicklungsumgebung* keine herausragende Bedeutung hat, könnte der Vergleich Luntbuild mit CruiseControl interessant sein, da CruiseControl im Gegensatz zu Luntbuild direkt über einen eigenen PlugIn Mechanismus erweiterbar ist. Es wäre zu klären, ob diese CruiseControl PlugIns eine über die durch verfügbare Maven2 PlugIns erreichbare Unterstützung hinausgehen, oder andere Funktionalitäten anbieten. Ansonsten bietet auch CruiseControl die Möglichkeit, über die Verwendung von Maven2 den gleichen Funktionsumfang bei der Montage zu bieten. Jedoch ist CruiseControl zur Zeit in seinen Steuerungsmöglichkeiten noch nicht so flexibel wie Luntbuild.
- CruiseControl ist im Wesentlichen aufgrund seiner Nicht-Integrierbarkeit in die *Entwicklungsumgebung* nicht verwendet worden. Durch die Bereitstellung eines CruiseControl PlugIns für Eclipse wäre die Auswahl nochmals zu überprüfen.

8.2 Mögliche Weiterentwicklungen

Die erreichte Lösung besteht aus einer Kombination von acht Produkten, die einzeln für sich weiter entwickelt werden. Dadurch erhöht sich der Verwaltungsaufwand um das System aktuell zu halten, da bei jeder neuen Version eines einzelnen Bestandteils überprüft werden muss, ob es neue nutzbare Verbesserungen gibt und ob eine Migration auf die neue Version die Funktionsfähigkeit der Gesamtlösung nicht beeinträchtigt. Wünschenswert wäre also eine Lösung, die aus einem Produkt besteht. Potentiell bietet Continuum hierfür eine gute Ausgangsbasis, da sowohl ANT, als auch Maven und Maven2 vom gleichen Hersteller stammen. Eine andere Möglichkeit wäre, wenn sich, ähnlich wie bei Linux, Distributoren fänden, die die einzelnen Bestandteile zu einer Gesamtinstallation verpackt anböten.

Literaturverzeichnis

Bücher:

(Preston 2007) PRESTON, W. Curtis: *Backup and Recovery*, 1. Aufl., O'Reilly Media, 2007, ISBN 0596102461

(Dittberner 2006) DITTBERNER, Fabian: *Unterstützung von continuous integration in verteilt entwickelnden Softwareprojekten*, Studienarbeit 2006, http://swt-www.informatik.uni-hamburg.de/publications/papers/Stud/Studienarbeit-FD.pdf

(Fowler 2002) FOWLER, Martin: *Enterprise Integration Patterns : Designing, Building, and Deploying Messaging Solutions,* Addison-Wesley Longman, November 2002, ISBN 0321200683

(Gamma 1994) GAMMA, Erich: *Design Patterns: Elements of Reusable Object-Oriented Software,* Addison-Wesley Professional Computing Series, 1994, ISBN 0201633612

(Züllighoven 2004) ZÜLLIGHOVEN, Heinz: *Object-Oriented Construction Handbook*, dpunkt.verlag, 2004, ISBN: 1-55860-687-4, http://www.dpunkt.de/buecher/2096.html

(Clark 2004) CLARK,Mike: *Pragmatic Project Automation, How to Build, Deploy, and Monitor Java Applications,* 3.Aufl, The Pragmatic Programmers, LLC., 2004, ISBN: 0-9745140-3-9, Extract Kapitel 3, http://www.pragmaticprogrammer.com/starter_kit/au/scheduled.pdf

(Mason 2004) MASON, Mike: *Pragmatic Version Control,Using Subversion*, 2.Aufl, The Pragmatic Programmers, LLC., 2006, ISBN 0-9776166-5-7

(Collins-Sussman 2004) Collins-Sussman, Ben, *Version Control with Subversion. Next Generation Open Source Version Control*, 1.Aufl., O'Reilly Media, 2004, ISBN 0596004486 & Ergänzte Onlineausgabe, http://svnbook.red-bean.com/

(Spillner 2005) SPILLNER, Andreas, LINT, Tilo: *Basiswissen Softwaretest*,3.Aufl, Dpunkt Verlag, 2005

(Pergande 2008) PERGANDE, Thorben: *Qualitätssicherung inkl. Programmierstilkontrolle mittels Maven2*, HAW-Hamburg, Bachelorarbeit 2008

(Raasch 2006) RAASCH, Jörg, *Skript zur Vorlesung „Softwareentwicklung“,* Wintersemester 05/06, HAW-Hamburg

(Link 2005) LINK, Johannes: *Softwaretests mit JUnit. Techniken der testgetriebenen Entwicklung,* 2.Aufl., dpunkt Verlag, 2005

(Zukunft 2006) ZUKUNFT, Olaf: *Skript zur Vorlesung „Datenbankdesign“,* Wintersemester 05/06, HAW-Hamburg

Normen:

[IEEE1061] Institute of Electrical and Electronics Engineers, IEEE Standard for a Software Quality Metrics

[ISO 9000] Deutsche Industrie Norm, Europäische Norm, International Standard Organisation, (DIN EN ISO 9000:2005) QM-Systeme – Grundlagen und Begriffe

[ISO 9126] ISO/IEC 9126:1991 Bewerten von Softwareprodukten, Qualitätsmerkmale und Leitfaden zu ihrer Verwendung (identisch mit DIN 66272, 94)

Webseiten:

Wenn nicht anders angegeben, Stand 18.1.2008

[AD] Microsoft Inc. 2008,
http://www.microsoft.com/germany/technet/datenbank/articles/600092.mspx

[ANTHILL] Urbancode, Inc. 2006, AnthillOS Website,
http://www.anthillpro.com/html/products/anthillos/default.html

[ANT] The Apache Software Foundation, THE APACHE ANT PROJECT,
http://ant.apache.org/

[APACHESERVER] The Apache Software Foundation, Apache HTTP SERVER PROJECT, http://httpd.apache.org/

[ARCHITEKTURMUSER] Arbeitskreises Architekturmuster in der GI Fachgruppe Software-Architektur, http://www.architekturmuster.de

[ASQFTest] ASQF - Arbeitskreis Software Qualität und Fortbildung e.V. SIG Software-Test
http://www.asqf.de/fachgruppen/software-test

[APACHEFOUNDATION]The Apache Software Foundation, 6.8. 2007, Apache ANT Website, http://www.apache.org

[CCCHANGELOG] http://sourceforge.net/project/showfiles.php?
group_id=23523&package_id=16338

[CONTINUUM] The Apache Software Foundation, 27.7. 2007,Apache Maven Continuum Website, http://maven.apache.org/continuum/index.html

[CRUISECONTROL] ThoughtWorks Inc, Alden Almagro, Paul Julius, 6.8.2007, CruiseControl framework Website, http://cruisecontrol.sourceforge.net/index.html

[CVS] Free Software Foundation, Inc., 1996, http://www.nongnu.org/cvs/

[Cygwin] http://www.cygwin.com/

[ECLIPSE] Eclipse Foundation, 6.8.2007, Eclipse Website, http://www.eclipse.org

[FOWLERCI] Martin Fowler, 1.5.2006, Continuous Integration,

http://www.martinfowler.com/articles/continuousIntegration.html

[GITest] Gesellschaft für Informatik e.V. - Fachgruppe TAV - Test, Analyse und Verifikation von Software, http://www.gm.fh-koeln.de/~winter/tav/

[GTB] GTB - German Testing Board e.V., http://www.german-testing-board.info

[GUMP] Apache foundation, 2.2.2007, Apache Gump Website, http://gump.apache.org/index.html

[IBM] Stelligent Incorporated, Paul Duvall, 5.9.2006, Automation for the people: Choosing a Continuous Integration server A survey of open source CI servers: CruiseControl, Luntbuild, and Continuum, http://www-128.ibm.com/developerworks/java/library/j-ap09056/index.html

[ISQI] International Software Quality Institute, http://www.isqi.org/

[JETTY] http://jetty.mortbay.org

[JSVN] http://www.alternatecomputing.com/jsvn/

[JUNIT] JUnit.org, http://www.junit.org/

[JWAM] http://www.c1-wps.de/leistungen/jwam/

[KDESVN] http://kdesvn.alwins-world.de/

[LDAP] http://tools.ietf.org/html/rfc4510

[LINUXDEAMON] http://www.ubuntu-forum.de/artikel/4115/Shellskript-als-Daemon-einrichten.html

[LUNTBUILD] Intland Software,6.8.2007, Luntbuild - automate and manage your builds, http://luntbuild.javaforge.com/

[LUNTCLIPSEINSTALL] http://luntbuild.javaforge.com/doc/luntclipse/index.html http://fisheye1.cenqua.com/browse/~raw,r=1.1/luntbuild/luntclipse/doc/index.html

[LUNTCLIPSE] Intland Software,6.8.2007, Luntclipse-PlugIn, http://luntbuild.javaforge.com/doc/luntclipse/index.html

[M2ECLIPSE] http://m2eclipse.codehouse.org

[MAVENGETTINGSTARTED] http://maven.apache.org/guides/getting-started/index.html

[MAVENUSER] Jason van Zyl, Jochen Kuhnle 4.7, 2007, Maven documentation, http://docs.codehaus.org/display/MAVENUSER/Home

[MYECLIPSE] Genuitec, LLC, myeclipse Webseite, http://www.myeclipseide.com/

[OMONDO] http://www.omondo.de/

[OOMetriken] F. Simon, D. Meyerhoff, OOMetriken zeigen große Qualitätspotenziale in komplexen Softwaresystemen, in: OBJEKTspektrum 06/02, http://www.sigs.de/publications/os/2002/06/simon_OS_06_02.pdf

[OPENSOUCEBUILDSYSTEMS] java-source.net, 6.8.2007, Open Source Build Systems in Java, http://www.java-source.net/open-source/build-systems

[PYTHON] http://www.python.org/

[SERVERMATRIX] The Codehaus, 2006, Continuous Integration Server Feature Matrix, http://damagecontrol.codehaus.org/Continuous+Integration+Server+Feature+Matrix

[SOURCEFORGE] http://sourceforge.net/

[SQUID] squid.cache.org, http://www.squid-cache.org

[SQ] Liggesmeyer, Peter: Software-Qualität: Spektrum, 2002

[SUBCLISPE] CollabNet, Inc. 2006, Subclipse-PlugIn Website, http://subclipse.tigris.org

[SUBVERSION] CollabNet, Inc. 2006, Subversion Website, http://subversion.tigris.org/

[StarTeam] http://www.borland.com/us/products/starteam/index.html

[SubclipseUseBranchTag] http://subclipse.tigris.org/branch_tag.html

[SubclipseUse] http://subclipse.tigris.org/screenshots.html

[UML] Bernd Oesterlich, Analyse und Design mit UML 2.1 (Regal)

[USELAB] http://users.informatik.haw-hamburg.de/~use-lab/

[VERSIONSKONTROLLSYSTEME] http://better-scm.berlios.de/comparison/comparison.html

[VisualSVN Server] VisualSVN Server http://www.VisualSVN.com

[WEBDAV] http://www.webdav.org/

[WINDIENSTE] http://www.serverhowto.de/Applikationen-als-Dienste-einrichten.228.0.html

Anhang

1 Installationsbeschreibungen

Inhaltsverzeichnis

1.1 Verwendete Betriebssysteme

Für die folgende Anleitung wurde eine Linuxdistribution als Serverbetriebssystem und WindowsXP als Betriebssystem für den Entwicklerclient gewählt. Beide Systeme wurden einmal als Server und einmal als Client verwendet, wodurch die Evaluation zeigt, dass es keine nennenswerten Unterschiede in der Wahl des Betriebssystems gibt. Einzige Ausnahme bildet die grafische Benutzungsschnittstelle für Subversion, die ausschließlich unter Windows betrachtet wurde. Der besseren Übersichtlichkeit halber wird nur die Kombination Linux Server / Windows Client dargestellt.

1.2 Unterschiede Linux / Windows

Ein Vorteil in der Wahl der genutzten Programme liegt darin, dass sie größtenteils in Java implementiert und somit plattformunabhängig sind. Sie verwenden keine in Java möglichen Zugriffe auf native Bibliotheken, so dass die Installation, Konfiguration und Nutzung i.d.R. unter beiden Betriebssystemen identisch sind. Lediglich die Orte für Konfigurationsdateien im Dateisystem sind unterschiedlich und die Darstellung der grafischen Benut-

zungsschnittstelle, sofern vorhanden, folgt dem Look&Feel des jeweiligen Systems.

Die Diensteverwaltung unter Windows und äquivalent die Daemonverwaltung unter Linux sind grundsätzlich unterschiedlich aufgebaut, so dass die Installation und in Betriebsnahme von Serverkomponenten als Dienste/Dämons sich unterscheiden. Sämtliche benutzten Serverkomponenten lassen sich als Einzelanwendung starten und betreiben, so dass die Nutzung als Dienst bzw. Dämon zunächst vernachlässigt werden kann.

Wie ein Dienst bzw. Daemon eingerichtet werden kann, findet sich unter [WINDIENSTE] bzw. [LINUXDEAMON].

Die Konzepte sowie sämtliche aus der *Entwicklungsumgebung* heraus genutzten Komponenten und die über Browser-Schnittstellen ausgelieferten Benutzungsschnittstellen bleiben von den durch das Betriebssystem bedingten Unterschieden unberührt.

1.3 Serversystem

Für die Bewertung der verschiedenen Softwarepakete wurde eine Testumgebung auf Basis der Ubuntu 7.10 – Distribution „Gutsy Gibbon“ verwendet. Hierbei handelt es sich zwar um eine Desktop-Variante des Betriebssystems, aber so konnte gleichzeitig die Integration in die *Entwicklungsumgebung* unter Linux in der selben Installation überprüft werden. Um den Installationsaufwand zu reduzieren und die Verteilungsmöglichkeiten der Softwarepakete evaluieren zu können, wurde eine Virtuelle Maschine genutzt, in der das Ubuntu in einem VMWare-Player auf einem Windows XP Hostsystem lief. Um den Aufwand für die Basisinstallation gering zu halten und die Nachvollziehbarkeit für den Leser zu erhöhen, wurde eine bereits verfügbare Basisinstallation genutzt, die kostenfrei unter http://www.vmware.com/appliances/directory/1063 zur Verfügung steht. Diese Basisinstallation liegt in deutscher Sprache vor und enthält neben dem fertig installierten und konfigurierten Ubuntu 7.10 auch die fertig konfigurierten VMWare-Tools, die benötigt werden, um z.B. Drag&Drop zwischen virtueller Maschine und Hostsystem zu ermöglichen.

Die folgenden Beschreibungen sollen einen schnellen Einstieg in die praktische Nutzung der in dieser Arbeit besprochenen Programme zur Integration von Build Management und Continuous Integration in Eclipse ermöglichen. Sie stellt keine vollständige Installations- und Konfigurationsanleitung dar. Insbesondere auf Sicherheitsaspekte wird hier nicht eingegangen. (Siehe zu Benutzerauthentifizierung auch Seite 52, 7.1 „Erreichter Stand“,Proxyanbindung.)

In der Virtuellen Maschine wurde zunächst das root-Passwort auf " " (Ein Leerzeichen) gesetzt. Das Standardpasswort ist "" (Ein null Zeichen langer String).

```
haw@jars-desktop:~$ su
Passwort:
root@jars-desktop:/home/haw# passwd
Geben Sie ein neues UNIX Passwort ein:
Geben Sie das neue UNIX Passwort erneut ein:
```

```
passwd: Kennwort erfolgreich geändert
root@jars-desktop:/home/haw#
```

Nach dem Einloggen (User "haw",Passwort "hawhaw") sollte man zunächst die vmware-toolbox als root starten, um z.B. Funktionen wie Copy & Paste zwischen Host- und Gast-System zu ermöglichen:

```
haw@jars-desktop:~$ su
Passwort:
root@jars-desktop:/home/haw# /usr/bin/vmware-toolbox &
[1] 6912
root@jars-desktop:/home/haw#
```

Das Toolbox-Fenster kann minimiert, sollte aber nicht geschlossen werden, da damit die gesamte Toolbox-Anwendung beendet würde.

Für die weitere Aktuallisierung des Systems wurden folgende in `/etc/apt/sources.-list` definierten *Repositories* verwendet:

```
deb http://archive.canonical.com/ubuntu gutsy partner
deb http://archive.ubuntu.com/ubuntu/ gutsy main universe multiverse restric-
ted
deb http://security.ubuntu.com/ubuntu/ gutsy-security universe main multiver-
se restricted
deb http://archive.ubuntu.com/ubuntu/ gutsy-updates universe main multiverse
restricted
```

Mittels apt-update und apt-upgrade wird das System zunächst auf den aktuellen Stand gebracht. Um apt-get nutzen zu können, muss eventuell ein Proxy eingestellt werden. Zum Überprüfen, welcher Proxy bereits eingestellt ist, verwendet man:

```
set | grep http_proxy
```

zum Setzen:

```
export http_proxy=http://username:password@IP:Port
```

wobei username, password, IP und Port als Platzhalter dienen und entsprechend zu ersetzen sind.

Für apt-upgrade ist, je nach verwendeter Hardware und Internetanbindung, mit deutlich über einer Stunde Bearbeitungszeit zu rechnen, in der auch die weiteren Installationsschritte nur teilweise vorgenommen werden können.

1.4 Entwicklersystem

Im WindowsXP Entwicklersystem wird die hosts-Datei (`c:\windows\system32\dri-vers\etc\hosts`) um den Eintrag "192.168.0.100 virtuelleMaschine" ergänzt, wobei

192.168.0.100 die IP-Adresse der virtuellen Maschine ist. Dadurch kann die Namensauflösung im Entwicklersystem ohne DNS-Server geschehen und im Browser über den URL http://virtuelleMaschine z.B. auf den Luntbuild-Apache der virtuellen Maschine zugegriffen werden. Ebenso werden später die Subversion-Zugriffe über diesen Hostnamen laufen können.

1.5 Installation & Basiskonfiguration der Software

Die im Folgendem beschriebene Installation folgt der Verteilung aus „Abbildung 5: Verteilungsdiagramm der verwendeten Anwendungen“.

Für die Prototypische Umsetzung wurde folgende Verzeichnisstruktur wurde gewählt. Die dargestellten Verzeichnisse sind dabei auf die in dieser Arbeit direkt besprochenen Verzeichnisse und Dateien reduziert.

1.5.1 Verzeichnisse Linux / Server

Gekürzte Darstellung des Verzeichnisses [server]

```
+---subversion.repositories
|  +---conf
|  |    passwd
|  |    authz
|  |    svnserve.conf
+---subversion
|  +---bin
|  |    svn.exe
|  |    svnadmin.exe
|  |    svnserve.exe
|  |    svnsync.exe
+---luntbuild-1.5.3
|  |  luntbuild-standalone.jar
|  +---logs
+---luntbuild.publish
|  \---FirstProject
|    \---FirstSchedule
|      +---luntbuild-2007-Dec-29.9
|      +---luntbuild-2007-Dec-29.10
|      +---luntbuild-2007-Dec-29.11
|      +---luntbuild-2007-Dec-29.12
|      +---luntbuild-2007-Dec-29.13
|      \---luntbuild-2007-Dec-29.14
|        |  revision_log.txt
|        |  revision_log.xml
|        |  revision_log.html
|        |  build_log.txt
|        |  build_log.xml
|        |  build_log.html
```

```
|         |
|         +---artifacts
|         +---jester_html_report
|         +---xref_html_report
|         +---jdepend_html_report
|         +---nunit_html_report
|         +---emma_html_report
|         +---cobertura_html_report
|         +---clover_html_report
|         +---pmd_html_report
|         +---junit_html_report
|         +---jcoverage_html_report
|         +---ncover_html_report
|         \---checkstyle_html_report
+---luntbuild.schedule.work
|   |  pom.xml
|   |
|   +---target
|   |   |  my-app-1.0-SNAPSHOT.jar
|   |   |
|   |   +---classes
|   |   |   \---com
|   |   |     \---mycompany
|   |   |       \---app
|   |   |           App.class
|   |   |           Level1a.class
|   |   |           Level1b.class
|   |   |           Level2a.class
|   |   |           Level3a.class
|   |   |           Level3b.class
|   |   |
|   |   +---test-classes
|   |   |   \---com
|   |   |     \---mycompany
|   |   |       \---app
|   |   |           AppTest.class
|   |   |           Level1aTest.class
|   |   |           Level1bTest.class
|   |   |           Level2aTest.class
|   |   |           Level3aTest.class
|   |   |           Level3bTest.class
|   |   |
|   |   \---surefire-reports
|   |       com.mycompany.app.AppTest.txt
|   |       TEST-com.mycompany.app.AppTest.xml
|   |
|   \---src
|     +---test
|     |   \---java
```

```
|     |     \---com
|     |       \---mycompany
|     |         \---app
|     |             AppTest.java
|     |             Level1aTest.java
|     |             Level1bTest.java
|     |             Level2aTest.java
|     |             Level3aTest.java
|     |             Level3bTest.java
|     |
|     \---main
|       \---java
|         \---com
|           \---mycompany
|             \---app
|                 App.java
|                 Level1aTest.java
|                 Level1bTest.java
|                 Level2aTest.java
|                 Level3aTest.java
|                 Level3bTest.java
|
\---Maven
  |
  +---conf
  |       settings.xml
  |
  +---bin
  |       m2.bat
  |       m2.conf
  |       mvn.bat
  |       mvnDebug.bat
  |       m2
  |       mvn
  |       mvnDebug
  |
  +---lib|
  +---boot
  +---initial.repository
  |   +---classworlds
  |   |   \---classworlds
  |   |       \---1.1-alpha-2
  |   |               classworlds-1.1-alpha-2.jar
  |   |               classworlds-1.1-alpha-2.jar.sha1
  |   |               classworlds-1.1-alpha-2.pom
  |   |               classworlds-1.1-alpha-2.pom.sha1
  |   |
  |   +---commons-beanutils
  |   |   \---commons-beanutils
```

```
|   |       +---1.6
|   |       |       commons-beanutils-1.6.pom
|   |       |       commons-beanutils-1.6.pom.sha1
|   |       |
|   |       \---1.7.0
|   |               commons-beanutils-1.7.0.jar
|   |               commons-beanutils-1.7.0.jar.sha1
|   |               commons-beanutils-1.7.0.pom
|   |               commons-beanutils-1.7.0.pom.sha1
|   |
|   +---commons-collections
|   |   \---commons-collections
|   |       +---2.0
|   |       |       commons-collections-2.0.jar
|   |       |       commons-collections-2.0.jar.sha1
|   |       |       commons-collections-2.0.pom
|   |       |       commons-collections-2.0.pom.sha1
|   |       |
|   |       +---2.1
|   |       |       commons-collections-2.1.pom
|   |       |       commons-collections-2.1.pom.sha1
|   |       |
|   |       +---3.1
|   |       |       commons-collections-3.1.pom
|   |       |       commons-collections-3.1.pom.sha1
|   |       |
|   |       \---3.2
|   |               commons-collections-3.2.jar
|   |               commons-collections-3.2.jar.sha1
|   |               commons-collections-3.2.pom
|   |               commons-collections-3.2.pom.sha1
.   .
.   .
.   .
|   |
|   \---com
|       \---mycompany
|           \---app
|               \---my-app
|                   |   maven-metadata-local.xml
|                   |
|                   \---1.0-SNAPSHOT
|                           my-app-1.0-SNAPSHOT.jar
|                           maven-metadata-local.xml
|                           my-app-1.0-SNAPSHOT.pom
|
\---repository
    +---commons-digester
    |   \---commons-digester
```

```
      .
      .
      .
      |
      +---junit
      |   \---junit
      |       +---3.8.1
      |       |       junit-3.8.1.pom.sha1
      |       |       junit-3.8.1.jar.sha1
      |       |       junit-3.8.1.pom
      |       |       junit-3.8.1.jar
      |       \---4.0.1
      |               junit-4.0.1.pom.sha1
      |               junit-4.0.1.jar.sha1
      |               junit-4.0.1.pom
      |               junit-4.0.1.jar
      .
      .
      .
      |
      \---com
          \---mycompany
              \---app
                  \---my-app
                      |   maven-metadata-local.xml
                      |
                      \---1.0-SNAPSHOT
                              my-app-1.0-SNAPSHOT.pom
                              my-app-1.0-SNAPSHOT.jar
                              maven-metadata-local.xml
```

1.5.2 Verzeichnisse Windows / Entwicklerclient

Gekürzte Darstellung des Verzeichnisses [client]

```
\---Dokumente und Einstellungen
  \---Benutzername
    \---.m2
        settings.xml

\---maven.project
  |  pom.xml
  |
  +---target
  |  |  my-app-1.0-SNAPSHOT.jar
  |  |
  |  +---classes
  |  |  \---com
  |  |    \---mycompany
```

```
|   |       \---app
|   |           App.class
|   |           Level1a.class
|   |           Level1b.class
|   |           Level2a.class
|   |           Level3a.class
|   |           Level3b.class
|   |
|   +---test-classes
|   |   \---com
|   |     \---mycompany
|   |       \---app
|   |           AppTest.class
|   |           Level1aTest.class
|   |           Level1bTest.class
|   |           Level2aTest.class
|   |           Level3aTest.class
|   |           Level3bTest.class
|   |
|   \---surefire-reports
|       com.mycompany.app.AppTest.txt
|       TEST-com.mycompany.app.AppTest.xml
|
\---src
    +---test
    |   \---java
    |     \---com
    |       \---mycompany
    |         \---app
    |             AppTest.java
    |             Level1aTest.java
    |             Level1bTest.java
    |             Level2aTest.java
    |             Level3aTest.java
    |             Level3bTest.java
    |
    \---main
      \---java
        \---com
          \---mycompany
            \---app
                App.java
                Level1a.java
                Level1b.java
                Level2a.java
                Level3a.java
                Level3b.java
```

```
\---Maven
  |
  +---conf
  |       settings.xml
  |
  +---bin
  |       m2.bat
  |       m2.conf
  |       mvn.bat
  |       mvnDebug.bat
  |       m2
  |       mvn
  |       mvnDebug
  |
  +---lib|
  +---boot
  +---repository                                    Synchronisiert mit Server
      +---commons-digester
      |   \---commons-digester
      .
      .
      .
      |
      +---junit
      |   \---junit
      |       +---3.8.1
      |       |       junit-3.8.1.pom.sha1
      |       |       junit-3.8.1.jar.sha1
      |       |       junit-3.8.1.pom
      |       |       junit-3.8.1.jar
      |       \---4.0.1
      |               junit-4.0.1.pom.sha1
      |               junit-4.0.1.jar.sha1
      |               junit-4.0.1.pom
      |               junit-4.0.1.jar
      .
      .
      .
```

1.5.3 Subversion

Die Installation von Subversion erfolgt unter Linux mittels:

```
apt-get install subversion
```

Im Anschluss wird ein Basisverzeichnis für ein Subversion-*Repository* mittels:

```
svnadmin create [server]subversion.repositories
```

erstellt. Unter Windows erfolgt die Installation durch einfaches Entpacken der Zip-Datei und anschließendes Hinzufügen des hierin enthaltenden bin-Verzeichnisses zur PATH-Variable. Die Erstellung eines *Repositories* erfolgt äquivalent zur Erzeugung unter Linux. Damit sich zunächst jeder ohne Authentifizierung mit dem *Repository* verbinden kann, muss in der Datei `[server]subversion.repositories/conf/svnserve.conf` der Eintrag

```
anon-access = write
```

ergänzt werden. Hiermit ist die Basiskonfiguration abgeschlossen. Es ist zwar möglich, Subversion auf Basis des Filesystems zu nutzen, da später aber die Möglichkeit geschaffen werden soll, von mehreren Client-Rechnern aus auf das *Repository* zuzugreifen, wird der in Subversion enthaltende Server genutzt.

1.5.4 Subversion-Server

Der Subversion-Server wird unter Linux mittels

```
svnserve -r [server]subversion.repositories -d --foreground --listen-host
192.168.0.100
```

gestartet. Unter Windows ist der Aufruf äquivalent. An dieser Stelle muss die IP Adresse angegeben werden, wenn sich im Netzwerk kein DNS-Server befindet, der "virtuelleMaschine" korrekt auflösen kann. Die IP Adresse, die die virtuelle Maschine benutzt, erhält man unter Linux mit ifconfig und unter Windows mit ipconfig /all.

1.5.5 Tortoise

Bei der Tortoise Installation durch den Installer sind keine relevanten Punkte zu beachten.

1.5.6 Repository vorbereiten

Im Filesystem wird ein Ordner "virtuelleMaschineSubversion.repository" angelegt, der die lokale Kopie des *Repositories* aufnehmen soll. Hier wird Tortoise die aus dem *Repository* entnommenen Dateien speichern und hier sollten selbst erstellte Dateien zum Hinzufügen in das *Repository* abgelegt werden.

Im Explorer wird mittels Rechtsklick->Tortoise->Import der Tortoise Import-Dialog geöffnet

und das *Repository* durch die Angabe des URL

```
svn://virtuelleMaschine
```

importiert.

1.5.7 Subclipse

Die Installation von Subclipse erfolgt unabhängig vom Betriebssystem in Eclipse. Unter http://subclipse.tigris.org/install.html findet sich eine detaillierte Installationsanleitung.

In der Anleitung wird angegeben, dass für die aktuellere Version:

```
Name: Subclipse 1.2.x (Eclipse 3.2+)
URL: http://subclipse.tigris.org/update_1.2.x
```

genutzt werden soll. Es kommt hierbei aber zu *Fehlern* bei der Auflösung von PlugIn-Abhängigkeiten. Nutzt man die Angaben für die ältere Version:

```
Name: Subclipse 1.0.x (Eclipse 3.0/3.1)
URL: http://subclipse.tigris.org/update_1.0.x
```

so wird ebenfalls die aktuelle Version 1.2.0 , jedoch ohne *Fehler* installiert. Nach Bestätigen der Lizenz ist Eclipse neu zu starten.

Zurück in Eclipse wird die Subversion-Perspektive geöffnet und ein neues *Repository* hinzugefügt. In diesem Fall

```
svn://vituelleMaschine
```

Die Nutzung von Subclipse ist unter [SubclipseUse] und [SubclipseUseBranchTag], unterstützt von Abbildungen ausführlich erklärt.

1.5.8 Maven2:

Die Maven2 Installation folgt der „Getting started with maven in 30 minutes“-Anleitung [MAVENGETTINGSTARTED] auf der Maven2 Homepage. Die dort angegebenen Schritte lauten:

```
Unix-based Operating Systems (Linux, Solaris and Mac OS X)
1. Extract the distribution archive to the directory you wish to install Ma-
ven 2.0.8. These instructions assume you chose /usr/local/apache-
maven-2.0.8 . The directory apache-maven-2.0.8 will be created from the ar-
chive.
2. Add the bin directory to your path, eg. export PATH=/usr/local/apache-ma-
ven-2.0.8/bin:$PATH
3. Make sure that JAVA_HOME is set to the location of your JDK, eg. export
JAVA_HOME=/usr/lib/jvm/java-6-sun-1.6.0.03
4. Run mvn --version to verify that it is correctly installed.

Windows 2000/XP
1. Unzip apache-maven-2.0.8-bin.zip to the directory you wish to install Ma-
ven 2.0.8. These instructions assume you chose C:\Program Files\Apache Soft-
ware Foundation\apache-maven-2.0.8
```

```
2. Add the bin directory to your path, by opening up the system properties
(WinKey + Pause), selecting the "Advanced" tab, and the "Environment Varia-
bles" button, then editing the PATH variable in the user variables. eg.
"C:\Program Files\Apache Software Foundation\apache-maven-2.0.8\bin";%PATH%
3. In the same dialog, make sure that JAVA_HOME is set to the location of
your JDK, eg. C:\Program Files\Java\jdk1.5.0_02
4. Run mvn --version to verify that it is correctly installed.
```

In der Installationsanleitung wird unter „How do I compile my application sources?“ erwähnt, dass beim ersten Kompilierungsvorgang die benötigten PlugIns und Bibliotheken heruntergeladen werden und dies einige Minuten dauern könnte. Während der Installationstests zu dieser Arbeit sind auch auf gut angebundenen PCs längere Wartezeiten von mehreren Stunden aufgetreten. So sind in einem Fall, bei dem 2MByte an Daten heruntergeladen wurden, über 6½ Stunden vergangen. In der Mehrzahl der Versuche reichte jedoch eine Zeit von 8 bis 12 Minuten.

Obwohl Maven im Betrieb sehr robust ist, könnte eine bessere Unterstützung für den Start auch dem erfahrenen Anwender die Arbeit deutlich erleichtern. So muss zur Zeit noch die Standardkonfigurationsdatei von Hand angelegt werden. Diese Datei kann minimal schon aus einem XML-„settings“-Eintrag bestehen, führt aber auf Grund ihres Fehlens zunächst zu einer Fehlermeldung. Daher hier gleich ein Beispiel der „settings.xml“ für die Konfiguration von Maven zur Nutzung über einen Proxy:

```
<settings>
     <proxies>
           <proxy>
                 <active>true</active>
                 <protocol>http</protocol>
                 <host>proxy.mycompany.com</host>
                 <port>8080</port>
                 <username>my-username</username>
                 <password>my-password</password>
           </proxy>
     </proxies>
</settings>
```

Für den ersten Start ohne Proxynutzung kann ebenfalls die obige settings.xml genutzt werden, es ist lediglich der <active>-Eintrag auf „false“ zu setzen.

Sowohl beim ersten Start von Maven als auch in dem Fall, dass PlugIns oder Bibliotheken im Maven-eigenen *Repository* nicht vorhanden sind, lädt Maven die benötigten Daten automatisch aus dem Internet nach. Dies kann zu großen zeitlichen Verzögerungen führen, und man gibt die Kontrolle darüber, welche *Softwarebestandteile* in welcher exakten Version bei der Montage verwendet werden, an Maven ab. Dies ist aus Gründen der Reproduzierbarkeit von Montagevorgängen abzulehnen. Maven bietet zwar die Möglichkeit, über das Maven-eigene *Repository* auch ältere Versionen von Bibliotheken zu verwenden, aber für die Bereitstellung einer homogenen Buildumgebung auf mehreren Rechnern bietet es sich an, das Maven-eigene *Repository* in einem Subversion *Repository* abzulegen und auf einem Rechner zentral auf dem gewünschten Stand zu halten. Hierbei wird dieses Subver-

sion *Repository* einzig von dem zentralem Rechner schreibend verwendet. Die lokalen Maven Installationen arbeiten im Offline-Modus, bei dem sie nicht selbstständig benötigte Daten aus dem Internet nachladen. Sie können jedoch auf das zentrale *Repository* zugreifen, das die jeweils für die *Entwickler* freigegebenen Bibliotheken zur Verfügung stellt. Die Abhängigkeit von der Internetanbindung wird dadurch ebenso verringert, was wiederum die Verfügbarkeit der einmal gespeicherten PlugIns und Bibliotheken sicherstellt. Durch die Verwendung einer weiteren, separaten Maven Installation kann eine separate Testumgebung für das Build Management geschaffen werden.

1.5.9 Maven-Repository zentral zur Verfügung stellen

Es soll die Möglichkeit geboten werden, von zentraler Stelle aus die auf den einzelnen Rechnern zur Verfügung stehenden Maven-Ropositories auf einem einheitlichen Stand zu halten. Außerdem soll es möglich sein, sowohl von der Konsole aus als auch aus Eclipse heraus Maven zu nutzen und dabei dasselbe *Repository* zu verwenden. Es bietet sich folgendes Vorgehen an:

Auf dem Server wird ein lokales Subversion *Repository* mit dem Namen „maven.repository“ erstellt.

```
svnadmin create [server]maven.repository
```

Es wird ein lokales Verzeichniss angelegt, in dem das initiale Maven *Repository* erstellt werden soll (Hier `[server]maven.initial.repository`).

Damit Maven dieses Verzeichniss nutzt, muss in der `settings.xml` der Eintrag

```
<localRepository>[server]maven.initial.repository</localRepository>
```

ergänzt werden. Wie unter [MAVENGETTINGSTARTED] beschrieben wird mittels

```
mvn archetype:create -DgroupId=com.mycompany.app -DartifactId=my-app
```

ein neues *Projekt* erstellt. Dabei lädt Maven die anfänglich benötigten Artefakte und PlugIns aus dem Internet nach und speichert sie im lokalen Maven *Repository*. Wann immer nun neue Module benötigt werden, sollten sie in dieses *Repository* eingespielt werden. Daher bietet es sich an, sämtliche *Projekte* mit allen benötigten *Montagezielen* einmal durch Maven montieren zu lassen. So werden sämtliche Abhängigkeiten aufgelöst und die benötigten Artefakte und PlugIns im initialen Maven *Repository* gespeichert. Insbesondere sollte auch die Dokumentationswebseite durch Maven erstellt werden.

```
mvn site
```

Hiernach steht ein fertig aufgebautes Maven *Repository* zur Verfügung, das mittels

```
svn import [server]maven.initial.repository file://[server]maven.repository
--message 'Initial repository layout'
```

in das vorbereitete Subversion *Repository* eingestellt wird. So ist es möglich, über svn

oder Tortoise dieses Subversion *Repository* auf einem Entwicklungsrechner zu nutzen.

Dazu wird in das lokale Maven Installationsverzeichnis das sich im Subversion *Repository* befindende Verzeichnis „repository“ ausgecheckt. Damit die lokale Maven Installation dieses *Repository* verwendet und im Offlinemodus arbeitet, müssen in der lokalen settings.xml die Einträge

```
<localRepository>[client]apache-maven-2.0.8/repository</localRepository>
<offline>true</offline>
```

gesetzt werden.

Bei der Evaluation ist es gehäuft zu Problemen beim ersten Auschecken gekommen, wenn in sehr kurzem zeitlichen Abstand nach dem Einstellen in das Subversion *Repository* bereits lokal ausgecheckt werden sollte. Nach einer Wartezeit von ca. 3 bis 4 Minuten tritt dieses Phänomen nicht auf.

Damit das *Repository* der lokalen Maven Installation aktuell bleibt, ist nun noch dafür Sorge zu tragen, dass das durch Subversion bereitgestellte *Repository* bei Bedarf oder regelmäßig aktualisiert wird. Dies wird leicht dadurch möglich, dass Luntbuild (mit Hilfe eines Subversion nutzenden Builders (siehe Seite 78 „Kopie erfolgreich montierter Komponenten in Repository“) bei jedem Montagevorgang zunächst einmal das *Repository* automatisch aktualisiert.

1.5.10 M2Eclipse-PlugIn:

Die m2eclipse Installation erfolgt nach dem gleichen Muster wie die Subclipse Installation. Die anzugebende URL lautet:

```
http://m2eclipse.codehaus.org/update/
```

In der Konsole sollte eine Fehlermeldung der Art

```
11.12.07 18:39:26 CET: [ERROR] Invalid user settings C:\Dokumente und Ein-
stellungen\Sascha.Kluth\.m2\settings.xml
11.12.07 18:39:26 CET: [ERROR] C:\Dokumente und
Einstellungen\Sascha.Kluth\.m2\settings.xml (Das System kann den angegebenen
Pfad nicht finden)
```

erscheinen. Dies liegt daran, dass die settings.xml, die m2eclipse nutzt, nicht automatisch erzeugt und auch nicht automatisch aus einer lokalen Maven Installation heraus kopiert wird. Damit es unabhängig davon, ob man die lokale Maven Installation über die Konsole benutzt, oder den Montagevorgang über das m2eclipse PlugIn aus der *Entwicklungsumgebung* heraus nutzt, immer zu den selben Montagevorgängen kommt, sollte die settings.xml der lokalen Maven Installation nun in das .m2 Verzeichnis kopiert werden. Das .m2 Verzeichnis befindet sich im Homeverzeichnis des aktuellen *Benutzers* und wird unter Windows mit dem Attribut „versteckt“ angelegt.

1.5.11 Luntbuild

Mittels

```
java -jar luntbuild-1.5.3-installer.jar
```

wird der grafische Installer von Luntbuild gestartet, der selbsterklärend durch die Installation führt. Im Anschluß kann der Luntbuild Server mittels

```
java -jar luntbuild-standalone.jar localhost 4242
```

direkt gestartet werden. Die weitere Konfiguration kann dann über das Webfrontend von Luntbuild erfolgen. Ein einfaches Angeben des Hosts und Ports zum Aufrufen des Webfrontend reicht nicht aus, da Luntbuild hier zwar eine Webseite ausliefert, aber keinen Link auf seine eigentliche Startseite anbietet. Der vollständige URL lautet:

http://localhost:4242/luntbuild

Beim Aufruf des URL ist man automatisch als anonymer *Benutzer* angemeldet und muss sich zunächst ausloggen, um sich als *Benutzer* „luntbuild“ mit dem Passwort „luntbuild“ anmelden zu können.

Um sich direkt einloggen zu können, kann man folgenden URL benutzen:

```
http://localhost:4242/luntbuild/luntbuild-login.html
```

Nun werden einige Basiseinstellungen vorgenommen: Unter Properties wird das Work Directory, das später das Luntbuild *Repository* aufnimmt und das Publish Directory, das später die fertig montierten Dateien aufnimmt, angegeben. Damit Luntbuild Informationen via eMail versenden kann, sind außerdem gültige SMTP-Daten anzugeben.

Des weiteren muss Luntbuild Benutzerinformationen mitgeteilt bekommen. [ANSATZPUNKT LDAP] Die Menüführung von Luntbuild hebt die Punkte für die Erstellung von *Benutzern*, *Projekten*, etc. leider nicht gut hervor. Sie sind jeweils oben rechts über das  Symbol zu erreichen.

Nun soll das bereits im Subversion *Repository* verfügbare Maven *Projekt* durch Luntbuild montiert werden. Dazu wird in Luntbuild unter dem Reiter „Projects“, oben rechts ein neues *Projekt* erstellt. Wichtig ist es, unter *Project admins, builders* und *viewers* einen vorhandenen *Benutzer* auszuwählen und unter *Notification methods* „eMail“ anzugeben. Beim Montieren eines *Projektes* führt Luntbuild zwar ein Log, aber falls es hierbei Probleme geben sollte, werden die Informationen ebenso als eMail versendet. Für ein ausführliches Log wird der Eintrag „verbose“ gewählt.

Das erstellte Projekt muss nun mit dem Subversion *Repository* verbunden werden. Dazu wird über den Reiter „VCS adaptors“ ein neuer Subversion Adapter konfiguriert. Die wichtigen Punkte sind hier die *Repository url base* (svn://virtuelleMaschine) und die *Quiet period*. Luntbuild wird regelmäßig im *Repository* nachschauen, ob es eine Modifikation gab. Damit nicht aus Eclipse herraus eine Modifikation eingespielt wird, die parallel von Lunt-

build bereits aus dem *Repository* gelesen wird, sollte hier ein Wert von 3-5 Minuten angegeben werden. Für die ersten Erprobungen reichen 60 Sekunden aus.

Der VCS Adapter funktioniert nicht ohne ein konfiguriertes Modul, für dessen Erstellung nun unten rechts das  Symbol zu erreichen ist. Es reicht dabei allerdings aus, das Modul ohne weitere Angaben zu speichern.

Nun ist Luntbuild in der Lage, Daten aus dem Subversion *Repository* zu lesen und eine lokale Kopie zu speichern. Diese Kopie soll nun montiert werden. Dazu muss ein Builder definiert werden. In diesem Fall ein Maven2Builder. Als *Command to run Maven2* wird das Maven2 Startskript „[server]apache-maven-2.0.8\bin\m2“ bzw. „[server]apache-maven-2.0.8\bin\mvn.bat“ angegeben. *Directory to run Maven2 in* wird auf [server]luntbuild.schedule.work gesetzt. *Goals to build* gibt das gewünschte Maven2 Ziel an. Also z.B. „package“ oder „site“. Die *Environment variables* werden dem direkt aus dem Beispiel unter dem Eingabefeld entnommen und lauten: „buildVersion=${build.version}“ und „scheduleName=${build.schedule.name}“ (getrennt durch einen Zeilenumbruch).

Um den Builder ausführen zu können muss nun noch abschließend ein Schedule eingerichtet werden. Dies ist auch notwendig, wenn man den Montagevorgang lediglich manuell starten möchte. Die Benutzerführung von Luntbuild ist an dieser Stelle nicht einheitlich. Zum Erstellungsdialog gelangt man oben rechts über das  Symbol. Die Build necessary condition wird zunächst auf „always“ gesetzt, damit überprüft werden kann, ob der Montagevorgang ausgeführt wird, noch unabhängig davon, ob sich an den Projektdateien etwas geändert hat. Als Associated builders und post-builders wird der bereits angelegte Maven2Builder ausgewählt und die Notify strategy auf „notify always“ gesetzt. Wichtig bei der Konfiguration ist, dass das *Directory to run Maven2 in* des Builders und das *Work subdirectory* des Schedules identisch sind, weil Maven sonst die Projektspezifische pom.xml nicht finden kann und der Montagevorgang fehlschlägt.

Nach dem Speichern kann mit dem  Symbol der Montagevorgang manuell aktiviert werden. Zuvor sollte noch mit dem 

Symbol die automatische Aktualisierung der Seite aktiviert werden. Nach einiger Zeit wird der Montagevorgang gestartet, ausgeführt und abgeschlossen. Sollte es hierbei zu Problemen kommen, finden sich Hinweise zu Problemen von Luntbuild im *system log* (oben rechts).

Probleme des Montagevorganges finden sich im build.log (z.B.: `[server]luntbuild.-publish/FirstProject/FirstSchedule/luntbuild-2007-Dec-29.5/build.log`).

Die montierten Daten (Programme, aber auch HTML-Seiten etc.) finden sich im Luntbuild Arbeitsverzeichnis (`[server]luntbuild.schedule.work/target`) und sollten von hier aus in ein Publish *Repository* kopiert werden. Luntbuild bietet hierfür noch keine direkte Unterstützung.

1.5.12 Kopie erfolgreich montierter Komponenten in Repository

Es ist möglich, eine Kopie der erfolgreich montierten *Komponenten* über ein Maven Target erstellen zu lassen. Da hier aber in das zentrale Subversion *Repository* gespeichert werden soll, wird diese Aufgabe direkt an Subversion mittels

```
svn mkdir -m "NewDir" svn://localhost/target
svn import svn://localhost/target -m "Initial import"
```

delegiert. Dazu bietet Luntbuild die Möglichkeit, in einem Schedule mehrere Builder auszuführen. Es werden also für beide obigen Kommandos je ein Builder vom Typ „Command builder“ eingerichtet. Durch das Ersetzten von „target“ durch „${build.version}“ wird sichergestellt, dass bei jedem Lauf ein neues Verzeichnis im Subversion *Repository* erstellt wird.

Backup Builds	
Builder type	Command builder
Build command	svn mkdir -m "${build.version}" svn://localhost/${build.version}
Run command in directory	
Wait for process to finish before continuing?	Yes
Environment variables	buildVersion=${build.version} scheduleName=${build.schedule.name}
Build success condition	result==0

Backup Builds2	
Builder type	Command builder
Build command	svn import svn://localhost/${build.version} -m "Initial import"
Run command in directory	
Wait for process to finish before continuing?	Yes
Environment variables	buildVersion=${build.version} scheduleName=${build.schedule.name}
Build success condition	result==0

Damit der Schedule die beiden Builder ausführt, müssen im Schedule noch die Einträge *Associated post-builders* auf „Backup Builds, Backup Builds2“ und *Post-build strategy* auf „post-build if success“ gesetzt werden. Durch den Eintrag „post-build if success“ wird verhindert, dass das Subversion *Repository* unnötig mit Dateien fehlgeschlagener Montagevorgänge belastet wird. Möchte man ausschließlich die montierten Projektbestandteile sichern, so ist im Builder „BackupBuilds2“ noch der Eintrag „Run command in directory“ auf das „target“-Verzeichnis zu setzen (`[server]luntbuild.schedule.work/target`).

Möchte man nicht für jede erfolgreiche Montage ein neues Verzeichnis im *Repository* anlegen, so wählt man einen festen Verzeichnisnamen. Das Verzeichnis sollte dann einmalig manuell im Subversion *Repository* angelegt werden. So kann auf die Ausführung des

Schedule „BackupBuild“ verzichtet werden. Subversion wird die Versionierung der Montagen selbst organisieren. Die Verwendung eines separaten Verzeichnisses pro Montage erlaubt dafür ein einfacheres Löschen einzelner Montagen nach einer gewissen Zeit. Weiteres zu Backupstrategien siehe (Preston 2007).

1.5.13 Luntclipse

Um nun nicht zwischen der *Entwicklungsumgebung* und dem Browser wechseln zu müssen, wird noch das Eclipse-PlugIn Luntclipse, wie unter [LUNTCLIPSEINSTALL] beschrieben, installiert und konfiguriert, so dass die Luntbuild Ansicht zur Verfügung steht und direkt verfolgt werden kann, welche Aktionen Luntbuild zur Zeit ausführt.

1.5.14 Luntbuild Continuous Integration

Luntbuild soll nun automatisch den Montagevorgang starten, wenn im Subversion *Repository* Änderungen eingespielt wurden. Dazu muss lediglich der Schedule rekonfiguriert werden. Setzt man den Eintrag "Repeat interval(minutes)" auf "3" und "Build necessary condition" auf "vcsModified or dependencyNewer" so wird Luntbuild alle 3 Minuten das Subversion *Repository* auf Projektänderungen prüfen und den Montagevorgang starten, falls Projektdateien oder Dateien anderer Projekte, von denen das eigentliche Projekt abhängt, geändert wurden. Dazu sollte einmal eine Projektdatei modifiziert, in das Subversion *Repository* eingespielt und der Vorgang in der Luntbuild Ansicht in Eclipse beobachtet werden.

Damit ist die Installation eines lauffähigen Build Management und Continuous Integration Systems abgeschlossen.

2 Ergebnisse der ABC-Analyse

In Tabelle 5: Gegenüberstellung Funktionalitäten ausgewählter Continuous Integration und Build Management Systeme mit den A-Anforderungen der ABC-Analyse werden die Ergebnisse der ABC-Analyse vollständig aufgelistet. Die Spalte A-Nr enthält dabei die jeweilige Nummer der Tabelle 2: Komprimierte Ergebnisse der ABC-Analyse.

Nr	A-Nr	Beschreibung	A	B	C
		Architektur			
		Das System sollte nutzbar sein als			
A	2	Einzelanwendung		B	
B	1	Client/Server	A		
C	3	OfflineModus verfügbar	A		
		Konfiguration			
		Die Konfiguration des Systems wird vom Administrator vorgenommen und sollte erfolgen via			
D		Datei			C
E		Wenn Datei, dann XML-Format		B	
F		Programminterface		B	
G		Webanwendung		B	
H		Schnittstellen z.B. SOAP			C
J		Kein Neustart erforderlich bei Rekonfiguration			C
		Defaultkonfiguration			
		Die Defaultkonfiguration ist für ein erstes Sichten des System relevant. Sie			
K		ist sofort nutzbar			C
L		ist nach geringen Modifikationen nutzbar		B	
		Proxyunterstützung			
		Die Anbindung erfolgt über gesicherte Zugänge, es wir eine Proxyunterstützung benötigt.			
M		für Versionskomponente		B	
N	4	für Buildkomponente	A		
		Konfigurationsumfang			
		Der Konfigurationsumfang bietet die Möglichkeit, das System an die Bedürfnisse der Nutzer anzupassen.			
O		viele Möglichkeiten		B	
P		geringe Komplexität		B	
Q		Zum ersten Start nötiger Konfigurationsumfang gering			C

Nr	A-Nr	Beschreibung	A	B	C
		Unterstützung unterschiedlicher Umgebungen (Entwicklung, Test, Integration, Deployment)			
		Das Gesamtsystem selbst sollte testbar sein und sich leicht transferieren lassen. Die Anpassung erfolgt			
R		via „%HOME%“			C
S		über Systemgrenzen hinweg Windows/Linux		B	
		Build Management			
T	5	*Die Montage soll automatisiert erfolgen, die Definition erfolgt in der Syntax von*			
U	6	ANT	A		
V		Maven			C
W	7	Selbständige Auflösung von Bibliotheksabhängigkeiten	A		
		Testframework			
		Für die Entwicklung von Java Programmen soll ein Testframework zur Verfügung stehen			
X	8	JUnit	A		
Y		Es soll möglich sein, spezielle Tests in einem separaten Kontext auszuführen.		B	
Z		Datenbanktests sollen optional nur ausgeführt werden, wenn die Datenbank zum Testzeitpunkt die notwendigen Ressourcen zur Verfügung stellen kann.		B	
		Verteilbarkeit			
2A		Die Verteilbarkeit des Systems ist erforderlich		B	
		Prozessunterstützung			
2B		*Die Unterstützung der Entwicklungsprozesschritte muss möglich sein*			
2C	1,2	Entwicklung>lokale Montage>lokaler Test>Integration>Integrationstest	A		
2D	1,2	Entwicklung>Integration>Integrationstest	A		
		Dokumentation			
		Eine Dokumentation (oder ein Hilfesystem) soll verfügbar sein.			
2E		hohe Quantität			C
2F		hohe Qualität		B	
2G		Vollständigkeit		B	
		Beispiele			
		Beispiele erleichtern das Erlernen des Systems.Gefortert			
2H		hohe Quantität			C
2J		hohe Qualität		B	
2K		Videos verfügbar			C

Nr	A-Nr	Beschreibung	A	B	C
		Demoprojekt			
		Ein Demoprojekt bietet einen leichteren Einstieg und sollte daher mitgeliefert werden.			
2L		wird automatisch generiert		B	
2M		aus IDE erstellbar		B	
2N		ohne weitere Anpassung erstellbar			C
2O		ohne weitere Anpassung testbar			C
2P		mit geringen Anpassungen erstellbar		B	
2Q		mit geringen Anpassungen testbar		B	
		Benutzungsschnittstelle Konfiguration			
		Um das System im Betrieb Warten zu können, muss es mindestens eine Benutzungsschnittstelle geben. Sie sollte ausgeprägt sein, als			
2R		Konsole			C
2S		Statische Webanwendung		B	
2T		Dynamische Webanwendung			C
2U		Programm			C
2V		Tray			C
2W	9	in IDE integriert	A		
		Benutzungsschnittstelle Nutzung			
		Um die Funktionen des Systems nutzen zu können, muss es mindestens eine Benutzungsschnittstelle geben. Sie sollte ausgeprägt sein, als			
2X		Konsole			C
2Y		Statische Webanwendung		B	
2Z		Dynamische Webanwendung		B	
3A		Programm			C
3B		Tray			C
3C	10	in IDE integriert	A		
		Projektmontage			
		Der Montageprozess kann aufwändig und langwierig sein.			
3D	11	kann gestoppt werden	A		
3E		kann nach Stopp fortgeführt werden		B	
3F		wird bei Rekonfiguration gestoppt und neu gestartet		B	
3G		wird bei Systemneustart automatisch neu gestartet		B	
3H	12	erfolgt automatisch bei Änderung	A		
3J		erfolgt per Intervall		B	
3K		erfolgt manuell		B	
3L		hohe Geschwindigkeit			C

Nr	A-Nr	Beschreibung	A	B	C
		Deployment			
		Das Montageergebniss soll später ausgeliefert werden als			
3M		JAR		B	
3N		WAR		B	
3O		ZIP			C
3P		FTP		B	
3Q	13	Klassen im Dateisystem	A		
		Fehlermeldungen			
		Wenn bei der Montage oder Integration Fehlerwirkungen auftreten, muss der Entwickler informiert werden. Je nach Güte sind sie unterschiedlich nutzbar.			
		Fehlermeldungen Buildprozess			
3R		Detailgrad hoch		B	
3S	14	Detailgrad mittel	A		
3T	14	Detailgrad gering	A		
3U	14	Qualität hoch	A		
		Fehlermeldungen System			
3V	15	Detailgrad hoch	A		
3W	15	Detailgrad mittel	A		
3X	15	Detailgrad gering	A		
3Y	15	Qualität hoch	A		
		Benutzerinformation via			
		Sowohl im Fehler- als auch im Erfolgsfall soll der Benutzer informiert werden. Und zwar über			
3Z		Webanwendung		B	
4A	16	Mail	A		
4B		SMS			C
4C	17	IDE	A		
4D		Programm			C
4E		RSS-Feed			C
4F		Services			C
4G	18	Benutzerinformation definierbar	A		
4H		nach Verantwortlichkeiten Fehlerklassen		B	
4J		nach Verantwortlichkeiten (Teil-)Projekten		B	
		Stabilität			
		Umgang mit Konfigurationsfehlern			
4K	19	Montageprozess wird frühzeitig beendet/nicht gestartet	A		
4L		System weiter nutzbar		B	
4M	20	System weiter erreichbar	A		

Nr	A-Nr	Beschreibung	A	B	C
		Konfigurationsverwaltung			
4N		Backup von Konfigurationen			C
4O		Konfigurationsvererbung		B	
		Erweiterbarkeit			
4P	21	Um das System zukünftigen Anforderungen anpassen zu können soll es erweiterbar sein	A		
4Q		Bereits eingesetzte, zu evaluierende oder zu entwickelnde Qualitätssicherungstools sollen in das System integriert werden können.		B	

Tabelle 6: Vollständige Aufstellung der Ergebnisse der ABC-Analyse

3 Ausschnitt aus dem Softwareentwicklungsprozess

Die Darstellung erfolgt exemplarisch. Wird nach dem *Test-First-Ansatz* entwickelt, so werden die Schritte E8-E17 vor den Schritten E2-E7 ausgeführt. Diverse Fehlermöglichkeiten und Bearbeitungen von Randbedingungen wurden zu Gunsten der Übersichtlichkeit weggelassen. Bei der Betrachtung der Integrationstests ab E27 wurde die Darstellung verkürzt. Auch die Integrationstest benötigen eine eigene Montageanleitung und unterliegen einem Testzyklus wie die Komponententests E10-E17. Die Darstellung erfolgt von oben nach unten in zeitlicher Abfolge. Sämtliche beschriebenen Schritte sind als potentiell optional zu betrachten. Welche Schritte ausgeführt werden, entscheidet der *Entwickler* an seinem *Expertenarbeitsplatz*.

Nr	Verantwortungsbereich Entwickler	Nr	Verantwortungsbereich Automat
E1	Spezifikation vom Designer übernehmen		
E2	Komponenten-Quelltext anfordern		
		A3	Versionskontrollsystem: Komponenten-Quelltext ermitteln und ausliefern
E4	Komponenten-Montageanleitung anfordern		
		A5	Versionskontrollsystem: Komponenten-Montageanleitung ermitteln und ausliefern
E6	Komponenten-Quelltext modifizieren und statisch testen/auf Spezifikation prüfen		
E7	Komponenten-Montageanleitung modifizieren		
E8	Komponententestquellen anfordern		
		A9	Versionskontrollsystem: Komponententestquellen ermitteln und ausliefern
E10	Komponententest-Montageanleitung anfordern		
		A11	Versionskontrollsystem: Komponententest-Montageanleitung ermitteln und ausliefern
E12	Komponententestquellen modifizieren		
E13	Montageanleitung Komponententest modifizieren		
E14	Komponententestquellen montieren		
E15	Komponententest testen		
E16	Testbericht auswerten		
E17	Test ergibt Fehler im Test: weiter bei E12 in Testmontageanleitung: weiter bei E13		
E18	Komponenten-Quelltext montieren, ergibt Komponente		
		A19	Testsystem: Komponente testen
E20	Testbericht auswerten		
E21	Test ergibt Fehler im Test: weiter bei E12 in Testmontageanleitung: weiter bei E13 in Komponente: weiter bei E6 in Komponenten-Montageanleitung: weiter bei E7 in Spezifikation: Spezifikationsänderung anfordern		
E22	Komponenten-Quelltext, Komponenten-Montageanleitung, Komponententestquellen und Komponententest-Montageanleitung übergeben		
		A23	Versionskontrollsystem:

				Übergebene Daten auf Konflikte prüfen
			A24	Versionskontrollsystem: Konflikt vorhanden, weiter bei E25 Kein Konflikt vorhanden: weiter bei A26
	E25	Konflikte lösen: weiter bei E22		
			A26	Versionskontrollsystem: Daten versionieren und speichern
	E27+	Integrationstestquellen anfordern		
	Tabelle gekürzt. Entfernte Schritte für Integrationstest entsprechen Komponententest E10-E17			
			A28	Versionskontrollsystem: Integrationstestquellen ermitteln und ausliefern
	E29	Komponente integrieren		
	E30	Integrationstest montieren		
	E31	Integrationstest ausführen		
	E32	Testbericht auswerten		
	E33	Test ergibt Fehler im Test: weiter bei E12 in Testmontageanleitung: weiter bei E13 in Komponente: weiter bei E6 in Komponenten-Montageanleitung: weiter bei E7 in Integrationstest: weiter bei E27+ in Integrationstest-Montageanleitung: weiter bei E27+ in anderer Komponente: Verantwortlichen für andere Komponente informieren. Dieser beginnt neuen Prozess mit E2 in Spezifikation: Spezifikationsänderung anfordern		
	E34	Integration erfolgreich. Prozess abschließen (Zeitdokumentation etc.).		

Tabelle 7: Ausschnitt aus dem Softwareentwicklungsprozess

4 Vergleichsübersicht Continuous Integration Server

Tabelle 8: Vergleichsübersicht Continuous Integration Server zeigt die unter http://damagecontrol.codehaus.org/Continuous+Integration+Server+Feature+Matrix am 8.12.2007 entnommen Spalten, der in dieser Arbeit besprochenen Continuous Integration Server.

Legend:
- The feature exists
- The feature exists, but is buggy or not well tested
- The feature is planned for the near future
- The feature doesn't exist

	CruiseControl	Anthill	LuntBuild	Gump	Continuum
Project info					
Project origin	ThoughtWorks	Urbancode	PMEase		Apache
Open Source	✓	✓	✓	✓	✓
Implementation language	Java	Java	Java	Java	Java
Free	✓	✓	✓	✓	✓
Online demo	here	✗			here
Number of **active** developers	5		3		4
SCM support					
CVS	✓	✓	✓		✓
Subversion	✓	✓	✓		✓
SCM related support					
SCM filtering	✓	✗	✗		
Multi-SCM	✓	✗	✓		
Can create new SCM repositories	✗	✗	✗		✗
Build management					
Parallel builds (ability to build several projects simultaneously)	✓	✗	✓		✓
Distributed builds	⚠	✗	✗		ℹ
Agents' code auto-updated		✗			
Manually force builds	✓	✓	✓		✓
SCM trigged builds	✓ (if supported by SCM)	✗	✓		✓
SCM poll based builds	✓	✓	✓		✓
Temporal build scheduling	✓	✓	✓		✓
Builds promotion	✗	✗	✓		ℹ
Interproject Dependencies		✓	✓		
Builds deletion	✗	✗	✓		ℹ
Reproduce history builds	✗	✗	✓		ℹ
Proactive (can prevent build breakages)	✗	✗			
Detect new failing tests while build		✗			
Notify when first test in build fails		✗			

	CruiseControl	Anthill	LuntBuild	Gump	Continuum
Security					
User authentication	✓	✗	✓	?	✓
User authorization schemes	✗	✗	✓	?	✓
LDAP Integration	✗	✗	✓	?	ℹ
Kerebos	?	✗	?	?	?
Single Sign On	?	✗	?	?	?
Custom JAAS	?	✗	?	?	?
RSA SecureID	?	✗	?	?	?
Publishing					
Email	✓	✓	✓	?	✓
Run executable	✓	✓	✓	?	✗
FTP	✓	✗	✗	?	✗
IRC	✗	✗	✗	?	✓
RSS	✓	✗	✓	?	ℹ
SCP	✓	✗	✗	?	✗
Windows System Tray	✓	✗	✓	?	✗
Formatted Logging	✓ XML	✗	✓ Xml, Html	?	✗
Yahoo Messenger	✗	✗	✗	✗	ℹ
MSN Messenger	?	✗	✓	?	✓
X10	✓	✗	✗	?	?
Web interface					
View changesets	✗	✓	✓	?	✓
Add new projects	⚠	✓	✓	?	✓
Clone projects	✗	✗	✓	?	✗
Delete projects	⚠	✓	✓	?	✓
Modify projects	⚠	✓	✓	?	✓
Kill builds	✓	✗	✓	?	ℹ
Pause builds	✓	✗	✓	?	ℹ
Access to build artifacts	✓	✓	✓	?	✓
Browse CI's working copy	✗	✗	✗	?	✓
Delete CI's working copy	✗	✗	✗	?	✓
Search in builds	✗	✗	✓	?	?
Historic graphs	✓	✗	✗	?	ℹ
Self-updating web page	✓	✓	✓	?	✓
Multi project support	✓	✓	✓	?	✓
Multi project view	✓	✓	✓	?	✓
Add/remove agent machines (for distributed builds)	?	✗	?	?	?

	CruiseControl	Anthill	LuntBuild	Gump	Continuum
Directly supported build tools					
Shell / command script	✓	✗	✓	?	✓
Ant	✓	✓	✓	?	✓
Groovy	✗	✗	✗	✗	✗
NAnt	✓	✗	✓	?	✓
Maven	✓	✗	✓	?	✓
Maven2	✓	✗	✓	✗	✓
Make	✗	✗	✗	?	✓
Tools integration					
ViewCVS	✗	✗	✓	?	i
Bugzilla	✗	✗	✗	?	
JUnit result rendering	✓	✗	✓	?	i
NUnit result rendering	✗	✗	✗	?	✗
Eclipse Plugin	✓	✗	✓	?	i
Installation and Configuration					
Windows installer	✗	✗	✓	?	i
Self contained distribution (except SCM clients)	✓	✗	✓	?	✓
Additional dependencies	JRE, SCM client	JDK, Servlet container, SCM client	jdk,tomcat,SCM client	?	JRE, SCM client
Execution platform	JVM	JVM	JVM	?	JVM
Project platform (what it can build)	Java + anything Ant/Maven/NAnt can build	Any language with supplied ant wrapper	Any language with supplied ant wrapper	?	Any language
Preferred build tool	Ant, Maven	Ant	Ant, Maven	?	Maven 2
Requires modifications to build scripts	NO	NO	NO	?	NO
Supports multiple projects	✓	✓	✓	?	✓
Automatic configuration from build script	✗	✗	✗	✗	✓ Maven only
Text file configuration	✓ XML	✗	✗	?	✗

Tabelle 8: Vergleichsübersicht Continuous Integration Server

5 Glossar

Administrator, Rolle
Eine Person, die für die Wartung und Pflege der Rechner Verantwortung trägt, auf dem das Versionskontrollsystem(VCS), Build Management und Continuous Integration Systemläuft.

Arbeitsplatz
Der Platz, an dem die zur Erledigung von Aufgaben benötigten *Werkzeuge* und *Materialien* zur Verfügung stehen. Siehe Seite 13, Kapitel 2.2.5.

Behälter
Datencontainer, Materialspeicher nach *WAM*. Siehe Seite 13, Kapitel 2.2.4.

Benutzer, Rolle
Kombination aus *Entwickler* und Tester.

Compilerdirektive
Ursprünglich Anweisungen im *Quelltext* zur Steuerung des Compilers, heute auch beim Compileraufruf angebbare Parameter zur Compilersteuerung.

Entwickler, Rolle
Eine Person, die Java-Projekte bearbeitet, Sourcen erstellt/modifiziert.

Entwicklungsumgebung
Summe der Programme, die der *Entwickler* für die Erledigung seiner Entwicklungsaufgaben nutzt. z.B. Eclipse

Expertenarbeitsplatz
Spezieller *Arbeitsplatz* für *Benutzer*, die exakt wissen, welche Aufgaben auf welchem Weg bearbeitbar sind. Siehe Seite 13, Kapitel 2.2.5.

FAQ
Frequently asked questions, Liste von gestellten und (i.d.R. auch) beantworteten Fragen

Fehler
Oberbegriff für *Fehlhandlung*, *Fehlerwirkung* und *Fehlerzustand*
Umgangssprachlich die nichterfüllung einer spezifizierten Anforderung

Fehlermaskierung
Durch das fehlerhafte Verhalten durch einen *Fehlerzustand* wird ein weiterer *Fehlerzustand* verdeckt.

Fehlerwirkung
Abweichung zwischen spezifiziertem und beobachtetem Verhalten.

Fehlerzustand
Inkorrektes Teilprogramm, das Ursache für eine *Fehlerwirkung* ist.

Fehlhandlung
Menschliche Handlung die zu einem *Fehlerzustand* führt.

GUI
Graphical user interface, Grafische Benutzungsschnittstelle einer Software.

IDE
Integrated developer environment, Integrierte *Entwicklungsumgebung*.

Integration
Zusammenführung von *Komponenten* zu einem Gesamtsystem.

Komponenten
Aus *Quelltexten* montierte Dateien.

Material
Zu bearbeitendes, passives Objekt nach *WAM*. Siehe Seite 13, Kapitel 2.2.3.

Metrik
Siehe *Softwaremaße*.

Montageziel
Durch den Montagevorgang erstellte Programmvariante (z.B. für unterschiedliche Zielplattformen, Kundengruppen etc.)

Projekt
Technische Zusammenfassung von *Komponenten* in der *Entwicklungsumgebung*.

Quelltext
Syntaktische Beschreibung einer *Komponente*.

Repository
Datencontainer, der durch ein Versionskontrollsystem zur Verfügung gestellt und verwaltet wird. Im Rahmen der WAM-Metaphern ein *Behälter*.

Scheduler
Programm zur zeitgesteuerten Ausführung (sich wiederholender) Vorgänge.

Softwarebestandteile
Dateien, wie *Quelltexte*, montierte *Komponenten*, aber auch Grafiken oder andere Mediadateien.

Softwaremaße
Funktion, die eine Softwareeinheit in einen Zahlenwert abbildet. Siehe Seite 18, Kapitel 2.7.

Testspezifikation
Definition, welches Testziel ein Test verfolgt und wie dies zu erreichen ist.

Test
Aus einer *Testspezifikation* abgeleitete Testquelltexte, die montiert vorliegen.

Test-First-Ansatz
Entwicklungsansatz bei dem zunächst die benötigten Testfälle ermittelt werden. Die eigentliche Software wird dann auf die Erfüllung der *Tests* hin entwickelt.

Testabdeckung
Ein Maß dafür, wie viele mögliche Anweisungen (oder Pfade, Entscheidungen) in einem Programmstück durchlaufen werden im Verhältnis zu den potentiell durchlaufbaren.

Tester, Rolle
Eine Person, die vorhandene *Quelltexte* mit *Tests* verbindet und *Tests* ausführt.

Werkzeug
Programm zur Bearbeitung eines *Materials* nach *WAM*. Siehe Seite 13, Kapitel 2.2.1.

6 Abbildungsverzeichnis

7 Tabellenverzeichnis

Versicherung über Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach §22(4) ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 1. Februar 2008 _____

Ort, Datum Unterschrift