bbv
MAKING VISIONS WORK.

BOOKLET

INTRODUCTION TO CONTINUOUS INTEGRATION

PROFITIEREN SIE VON UNSERER ERFAHRUNG!

Kontakt Schweiz
bbv Software Services AG
Blumenrain 10
6002 Luzern
Telefon: +41 41 429 01 11
E-Mail: info@bbv.ch

Kontakt Deutschland
bbv Software Services GmbH
Agnes-Pockels-Bogen 1
80992 München
Telefon: +49 89 452 438 30
E-Mail: info@bbv.eu

INHALT

1 EINLEITUNG

Die Fähigkeit, qualitativ hochstehende Software zu entwickeln und dabei stetig auf sich ändernde Anforderungen reagieren zu können, zeichnet ein gutes Software-Entwicklungsteam aus. Agile Methoden wie Scrum und Test Driven Development haben genau dies zum Ziel. Continuous Integration ist ein Teil eines Entwicklungsprozesses, der sich nahtlos in die zuvor genannten Methoden einbetten lässt, um die hohe Qualität der Software zu ermöglichen.

Dieses Booklet klärt über Absichten, die mit Continuous Integration verfolgt werden (Kapitel 3), auf. Es zeigt die Voraussetzungen, die zur Umsetzung erfüllt sein müssen (Kapitel 3), und veranschaulicht, wie eine beispielhafte Continuous-Integration-Umgebung aufgebaut ist (Kapitel 4).

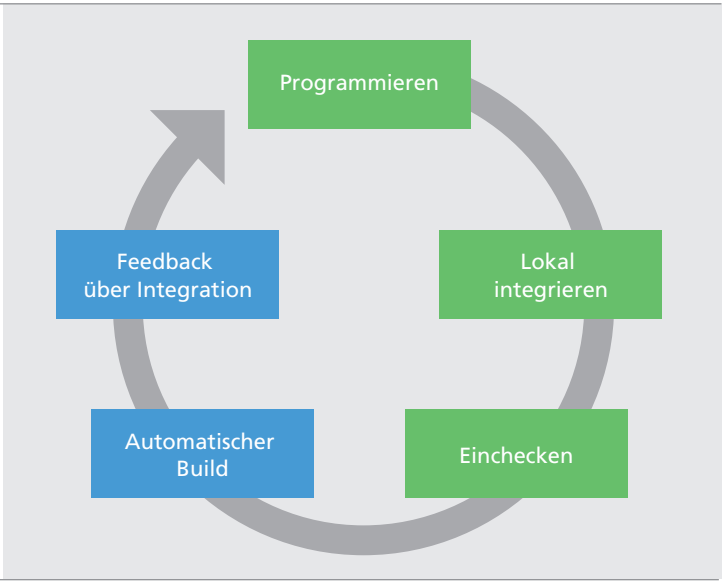
Im Booklet wird durchgehend die männliche Form für Projektleiter und Entwickler benutzt, es sei hier aber ausdrücklich darauf hingewiesen, dass immer auch die Projektleiterinnen und Entwicklerinnen angesprochen werden.

2 ÜBERBLICK

Martin Fowler (2006) schreibt im Artikel über Continuous Integration:
«Continuous Integration is a software development practice where members of a team integrate their work frequently, usually each person integrates at least daily – leading to multiple integrations per day. Each integration is verified by an automated build (including test) to detect integration errors as quickly as possible. Many teams find that this approach leads to significantly reduced integration problems and allows a team to develop cohesive software more rapidly.»

Der Entwicklungsprozess mit Continuous Integration lässt sich als immer wiederkehrender Zyklus darstellen.

Abbildung 1: Continuous-Integration-Zyklus



Programmieren, lokal integrieren und einchecken werden vom Entwickler auf seinem Rechner durchgeführt. Automatischer Build (dazu gehören Kompilieren, Unit-Tests, Code-Analyse etc.) und Feedback über Integration werden von der Continuous-Integration-Umgebung automatisch durchgeführt, nachdem der Entwickler seine Änderungen in die Sourcecode-Verwaltung übertragen hat.

3 NUTZEN UND LÖSUNGEN

In diesem Kapitel erläutere ich Ihnen die Ziele, die mit der Einführung und Anwendung von Continuous Integration erreicht werden können. Wenn sich eines oder mehrere der Ziele aus der Liste mit Ihren Zielen decken, dann ist Continuous Integration die Methodik, die Ihnen das Erreichen Ihrer Ziele erleichtern kann. bbv Software Services AG besitzt das Know-how, um Sie bei der Einführung und Umsetzung von Continuous Integration zu beraten und zu unterstützen.

Das übergeordnete Ziel von Continuous Integration besteht darin, einen professionellen Software-Entwicklungszyklus zu erhalten. Dieses sehr anspruchsvolle Ziel ist nicht mit einem Schritt zu erreichen. Für den Einstieg in die Continuous-Integration-Thematik ist es einfacher, sich zunächst mit kleineren Zwischenzielen auseinanderzusetzen, um darauf aufbauend dem übergeordneten Ziel eines professionellen Software-Entwicklungszyklus näherzukommen.

Mit Continuous Integration ergeben sich für den Auftraggeber einige Vorteile:

- Bessere Lenkungs- und Controlling-Möglichkeiten
- Höhere Qualität
- Sicherheit in der Zusammenarbeit

Können Sie in Ihrem aktuellen Projekt mit einem Doppelklick aus dem Sourcecode eine lauffähige Version Ihres Systems erzeugen (builden)? Dauert Ihr Build-Prozess nur wenige, vielleicht maximal fünf Minuten? Wenn Sie diese beiden Fragen mit «Ja» beantworten können, dann haben Sie bereits zwei Zwischenziele auf dem Weg zu Continuous Integration erfüllt. Beantworten Sie eine oder beide Fragen mit «Nein», dann haben Sie bereits ein oder zwei anspruchsvolle Zwischenziele, die es zu erreichen gilt, um mit Continuous Integration durchzustarten.

Nun wollen wir gemeinsam wichtige Zwischenziele formulieren, deren Bedeutung für den Continuous-Integration-Prozess bestimmen und allgemeine Lösungsansätze erläutern. Konkrete Vorschläge für den Einsatz von Werkzeugen folgen später in diesem Booklet. Auch die ersten beiden Zwischenziele sind Teil dieser Betrachtung.

Wir identifizieren folgende Zwischenziele:

Ziel	Priorität
Zuverlässiger, reproduzierbarer Build-Prozess	1
Fehler schnell finden und beheben	2
Schneller Build-Prozess	3
Toolset	4
Vorlagen und Anleitungen	5
Automatisiertes Deployment	6

3.1 ZUVERLÄSSIGER, REPRODUZIERBARER BUILD-PROZESS

Basierend auf einem bestimmten Stand des Quellcodes kann die Software immer wieder neu gebuildet werden, ohne dass sich deren Verhalten auch nur geringfügig verändert. Zu jedem Zeitpunkt im Projekt kann ein beliebiger älterer Stand des Quellcodes genommen und wieder gebuildet werden. Die Software verhält sich genauso, wie sie sich verhielt, als der gewählte Quellcode der aktuellste war.

Zuverlässig ist der Build-Prozess dann, wenn ein Entwickler davon ausgehen kann, dass der Build fehlerfrei durchläuft. Ein Build-Prozess, bei dem der Entwickler regelmässig manuelle Tätigkeiten ausführen muss, ist nicht zuverlässig.

Wir haben gesehen, was dieses erste Ziel bedeutet. Aber wie kann es erfüllt werden? Mit einem Version-Control-System (VCS) erreichen wir, dass zu jedem Zeitpunkt auf die aktuelle oder eine beliebige frühere Version des Quellcodes zugegriffen werden kann. Ein Grundsatz im Umgang mit einem VCS ist, dass jeder Stand funktioniert. Um dies

sicherzustellen, muss der Build-Prozess auch vor jedem Commit[1] lokal mit dem aktuellen Stand des VCS durchgeführt werden.

Damit der Build in jeder Version reproduzierbar ist, gehören auch die Build-Scripts unter VCS. Diese müssen genau wie Quellcode behandelt werden. Was bedeutet, dass sie genau wie der übrige Quellcode verbessert, bei Änderungen entsprechend angepasst und Fehler darin korrigiert werden.

Der Build-Prozess bei Continous Integration beinhaltet nicht nur das Erstellen eines ausführbaren Programms aus dem Quellcode, sondern auch das Ausführen von automatisierten Tests. Man spricht auch von selftesting code. So kann sichergestellt werden, dass das Programm nicht nur erstellt und gestartet werden kann, sondern sich auch korrekt verhält.

3.2 FEHLER SCHNELL FINDEN UND BEHEBEN

Warum ist es ausgesprochen wichtig, Fehler so früh wie möglich zu finden? Je eher ein Fehler gefunden und behoben wird, desto weniger Kosten werden verursacht. Entsprechend teurer wird ein Fehler, je später er entdeckt und behoben wird.

Einen Fehler findet man am schnellsten, wenn man die Software regelmässig intensiv testet. Dies kann am effizientesten mit vollautomatisierten Tests gemacht werden, welche am besten mit einer speziellen Testing Library entwickelt werden.

Grundsätzlich sollten diese Tests nach jeder Änderung im VCS erfolgreich durchgeführt werden. Dazu ist ein spezieller Continuous-

[1] Oft auch als «einchecken» bezeichnet.

Integration-Server die beste Lösung. Dieser überwacht das VCS und führt bei jeder Änderung den vollständigen Build-Prozess[2] durch und gibt entsprechendes Feedback über Erfolg oder Misserfolg.

3.3 SCHNELLER BUILD-PROZESS

Weshalb muss der Build-Prozess auch noch schnell sein? Ein Grund ist, Fehler schnell finden und beheben (Kapitel 3.2). Der andere wichtige Grund sind die Entwickler. Dauert der Build-Prozess zu lange, sinkt die Motivation, ihn vor jedem Commit durchzuführen, was dazu führt, dass sich vermehrt Fehler einschleichen, die früher hätten gefunden werden können. Der lokale Build-Prozess soll nur wenige Minuten in Anspruch nehmen.

Nicht immer ist es mit herkömmlichen Mitteln möglich, die Build-Zeit auf eine annehmbare Zeit zu reduzieren. In solchen Fällen gibt es einige Möglichkeiten zur Optimierung. Einerseits kann das System in Subsysteme unterteilt werden, sodass nicht bei jeder Änderung das komplette System neu gebuildet werden muss. Eine zweite Möglichkeit ist die verteilte Kompilation mit einem entsprechenden Tool. Als weitere Variante kann sogenanntes Staged Building eingesetzt werden, bei dem der Build-Prozess in mehreren Stufen durchgeführt wird. Welche Variante oder eine Kombination mehrerer Varianten eingesetzt wird, muss individuell beurteilt werden.

3.4 TOOLSET

Eine kontinuierlich verbesserte Sammlung an Tools, die in den jeweils aktuellsten Versionen eingesetzt werden, hilft dem Entwicklungsteam die zur Verfügung stehende Zeit optimal für produktive Arbeiten zu nutzen. Tools sollen nicht ständig ausgetauscht werden, da jedes Tool

[2] Zum vollständigen Build-Prozess gehört nebst Kompilieren zumindest auch das Ausführen der Testsuiten.

auch eine Lernkurve mit sich bringt. Je besser ein Entwickler seine Tools kennt, desto gewinnbringender kann er sie auch einsetzen. Zu den wichtigsten Tools gehören[3]: Build Tool, Bibliotheken für Unit- und System-Tests, VCS, Continuous-Integration-Umgebung (CIE), Dokumentationstools, Code-Bibliotheken, IDE Plugins, welche dem Entwickler wiederkehrende Aufgaben abnehmen oder ihn dabei unterstützen (Refactoring, Unit-Tests in der IDE ausführen etc.). Diese Liste kann je nach Projekt und Grösse des Teams beliebig erweitert werden.

3.5 VORLAGEN UND ANLEITUNGEN

Warum soll ein Build-Script für jedes Projekt neu entwickelt werden? Hat der Entwickler eine Vorlage, die er für das neue Projekt anpassen kann, spart er schon zu Beginn des Projekts wertvolle Zeit. Weshalb soll sich ein Entwickler bei jedem Projekt neu überlegen, wie er die Struktur im VCS gestalten soll? Ist von den Erfahrungen der ganzen Entwicklungsabteilung zentral dokumentiert, wie eine Struktur aussehen soll, ist wieder Zeit gespart.

Vorlagen dienen den Entwicklungsteams vor allem längerfristig. Für einige Aufgaben sind Vorlagen kaum umsetzbar, weshalb die entsprechenden Vorgehensweisen am einfachsten an einem zentralen, allen Entwicklern zugänglichen Ort als kurze Anleitungen abgelegt werden sollen. Eine Online-Wikisoftware oder ein Dokumentenverwaltungssystem sind zwei geeignete Kandidaten dazu. Script Templates können auch in einem gemeinsamen Tools Repository verwaltet werden.

Vorlagen sparen insbesondere zu Beginn eines Projektes viel wertvolle Zeit, und Anleitungen wie auch Vorlagen sind im späteren Verlauf des Projekts immer wieder hilfreich.

[3] Die Reihenfolge macht keine Aussage über eine Priorisierung.

3.6 AUTOMATISIERTES DEPLOYMENT

Stehen regelmässig Releases für den Kunden oder auch zu Testzwecken an, muss in jedem Zyklus einmal das ganze System verteilt[4] werden. Jedes Mal muss dafür kostbare Zeit aufgewendet werden. Gibt es ein automatisiertes Deployment (z. B. mittels Script) kann diese Zeit gespart und für die Entwicklung zusätzlicher Funktionalität, Behebung von Fehlern oder zur Stabilisierung der Software aufgewendet werden. Wichtig für ein effizient funktionierendes automatisiertes Deployment ist eine kontinuierliche Anpassung an die Bedürfnisse des Systems. Bereits früh im Projekt, auch wenn das System noch klein und übersichtlich ist, muss das automatisierte Deployment angepackt werden. Wenn man erst bei der Überschreitung einer gewissen Komplexität mit dem automatisierten Deployment beginnt, droht die Gefahr, dass der notwendige Initialaufwand zu gross ist und das automatisierte Deployment fallen gelassen wird. Entwickelt man das automatisierte Deployment kontinuierlich weiter und passt es dem wachsenden System an, ist der Aufwand über die ganze Dauer des Projekts verteilt und kann bei der Planung und Kalkulation berücksichtigt werden.

3.7 VORAUSSETZUNGEN

Es gibt einige komplett unterschiedliche Aspekte, die innerhalb einer Entwicklungsabteilung oder eines Entwicklungsteams erfüllt sein müssen, um Continuous Integration zu Ihrem Vorteil zu nutzen. Nicht nur technische Aspekte wie Hardware-Infrastruktur und Software sind entscheidend für eine erfolgreiche Umsetzung, sondern auch menschliche Herausforderungen sind zu meistern. Zu den drei genannten Aspekten erläutere ich jeweils einige wichtige Voraussetzungen.

[4] Deployed

Damit sich Continuous Integration auszahlt, muss das Projekt eine gewisse minimale Komplexität aufweisen. Sehr kleine «Einzeiler»-Projekte profitieren von Continuous Integration nicht, da der Initialaufwand zu gross sein kann. Existiert hingegen bereits eine Infrastruktur, kann sich Continuous Integration auch für kleinere Projekte auszahlen.

3.8 MENSCHLICHE VORAUSSETZUNGEN

Eine der wichtigsten Voraussetzungen für erfolgreiche Softwareentwicklung ist ein gut funktionierendes Team, in dem offen kommuniziert wird. Alle Mitglieder müssen miteinander sprechen können und wollen. Wie Kent Beck (1999) in Extreme Programming Explained manifestierte, gehört der Code dem ganzen Team und jeder Entwickler kann zu jeder Zeit beliebigen Code ändern. Dadurch kann rasch auf Defekte, Änderungen der Anforderungen und sonstige sich verändernde Umstände reagiert werden. Dies bringt aber auch mit sich, dass alle Entwickler für allen Code verantwortlich sind, egal welches Teammitglied den ursprünglichen Code geschrieben hat.

Die Disziplin jedes einzelnen Entwicklers, aber auch die Disziplin als Team ist dazu von grosser Bedeutung. Ein gefundener Fehler muss so schnell wie möglich korrigiert werden. Die Gründe dafür haben wir bereits im Kapitel 3.2 Fehler schnell finden und beheben kennengelernt. Lässt die Disziplin des Teams nach, werden Fehler erst später oder möglicherweise gar nicht korrigiert und verursachen höhere Kosten, als wenn der Fehler früher behoben worden wäre. Arbeitet das ganze Team weniger diszipliniert, kann der technische Projektleiter das Team darauf aufmerksam machen, um die Mitglieder wieder stärker zu sensibilisieren. Ein einzelner Entwickler, der weniger diszipliniert handelt, wird von einem gut funktionierenden Team fast automatisch wieder auf den Pfad der Tugend zurückgebracht.

3.9 VORAUSSETZUNGEN AN DIE INFRASTRUKTUR

Die Continuous-Integration-Umgebung sollte einfach ins bestehende Firmennetzwerk integriert werden können. Sollen das VCS oder der Continuous-Integration-Server einmal vom Internet aus zugreifbar sein, empfiehlt es sich, diese bereits zu Beginn in der DMZ zu platzieren.

Das Firmennetzwerk muss stabil, performant und ausfallsicher sein, damit einerseits für die Entwickler keine unnötigen Wartezeiten beim Committen entstehen und andererseits die CIE jederzeit den aktuellen Quelltext abholen und den Build-Prozess durchführen kann. So ist das Team fortlaufend über den Integrationszustand des Projekts informiert.

3.10 VORAUSSETZUNGEN AN DIE SOFTWARE

In der Continuous-Integration-Umgebung wird eine Continuous-Integration-Serversoftware betrieben, die das Management des Prozesses übernimmt. Weiter werden ein Build Tool, das den definierten Build-Prozess durchführt, und Testing Frameworks, mit denen die unterschiedlichen Klassen von Tests durchgeführt werden können, eingesetzt. Optional können weitere Tools zur detaillierten Analyse der Software verwendet werden.

4 CONTINUOUS-INTEGRATION-UMGEBUNG (CIE)

In diesem Kapitel bauen wir gemeinsam eine skalierbare Continuous-Integration-Umgebung mit konkreten Tools auf. Im ersten Schritt bauen wir als solide Grundlage eine Basiskonfiguration. Im zweiten Schritt werden wir die Umgebung ausbauen, um mehrere Projekte in der Umgebung am Laufen zu haben.

4.1 BASISKONFIGURATION

Zum Starten benötigen wir folgende Server-Infrastruktur:

- Version-Control-System (VCS)-Server
- Continuous-Integration-Server
- Build Agent

Um die CIE in Zukunft kostengünstig und schnell skalieren zu können, setzen wir nur einen physikalischen Server, den Host, ein. Auf diesem läuft ein Virtualisierungssystem, und darin werden die drei oben genannten Systeme in einer jeweils eigenen virtuellen Maschine installiert.

Als Virtualisierungssystem empfehle ich für den Einstieg VMware vSphere Hypervisor™[5]. Dieses Produkt kann alles, was für eine virtualisierte Umgebung notwendig ist, ist ausserdem gratis und kann später problemlos auf eine kostenpflichtige Version mit mehr Möglichkeiten und Unterstützung für grössere Rechenpower migriert werden.

Damit in der virtualisierten Umgebung genügend Rechenpower und ein wenig Leistungsreserven vorhanden sind, empfehle ich folgende minimalen Anforderungen an die Hardware des Hosts:

- 1 64-Bit CPU mit 4 Cores
- 12 GB RAM
- 500 GB Harddisk

Für die virtuellen Maschinen empfehle ich folgende Konfigurationen:

VCS-Server

- Windows-Server 2008 R2[6]
- Visual-SVN-Server Standard-Edition[7]
- Mind. 1 GB RAM zuweisen

Die Standard-Edition des Visual-SVN-Servers ist kostenlos und verfügt über alle notwendigen Features. Ein kostenpflichtiges Upgrade auf die Enterprise Edition wird für eine Active-Directory-Anbindung und eine umfangreichere Administration des Servers benötigt.

Continuous-Integration-Server

- Windows-Server 2008 R2[6]
- JetBrains TeamCity[8]
- Mind. 2 GB RAM zuweisen

Die Professional Edition des TeamCity-Servers ist ebenfalls kostenlos und verfügt über alle notwendigen Features, ist aber auf 20 Buildkonfigurationen beschränkt. Für zusätzliche Buildkonfigurationen und ein umfangreicheres Berechtigungssystem mit Active-Directory-Anbindung wird ein Upgrade auf die kostenpflichtige Enterprise Edition benötigt.

Build Agent

- Windows 7 Professional 64-Bit
- TeamCity-Build-Agent-Software[9]
- Mind. 3 GB RAM zuweisen

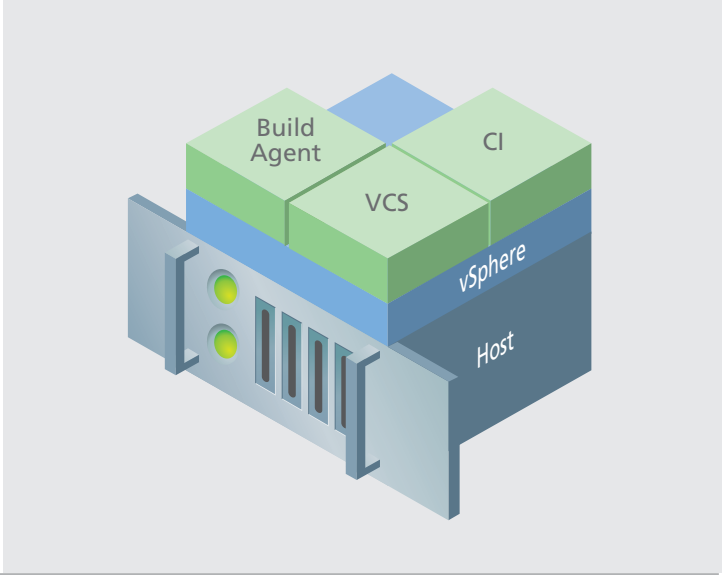
Sowohl bei der Professional als auch bei der Enterprise Edition des TeamCity-Servers sind drei Build Agents ohne zusätzliche Lizenzkosten inbegriffen. Unabhängig davon, welche Edition Sie einset-

[5] http://www.vmware.com/products/vsphere-hypervisor/overview.html
[6] Um Lizenzkosten zu sparen, wäre auch Windows 7 Professional 64-Bit möglich. Soll der Server aber auch ausserhalb des Intranets zugänglich sein, empfehle ich, aufgrund der besseren Sicherheit unbedingt einen Windows-Server 2008 R2 einzusetzen.
[7] http://www.visualsvn.com/server/
[8] http://www.jetbrains.com/teamcity/
[9] Wird mit der TeamCity-Server-Installation mitgeliefert.

zen, können Lizenzen für zusätzliche Build Agents erworben werden. Mit dieser Konfiguration der virtuellen Maschinen bleiben genügend Ressourcen übrig, um später zwei weitere Build Agents betreiben zu können, ohne dass die Hardware aufgerüstet werden muss.

Abbildung 2 veranschaulicht unsere virtualisierte Umgebung.

Abbildung 2:
Virtualisierte Umgebung



Auf dem Host läuft der vSphere Hypervisor, der die Hardware virtualisiert, und darin laufen die drei virtuellen Maschinen mit den jeweiligen Diensten.

4.2 ERWEITERTE KONFIGURATION

Wird das Projekt grösser, gibt es neue Projekte oder stellt ein Projekt besondere Anforderungen an die Continuous-Integration-Umgebung, wie beispielsweise automatisierte UI-Tests, können die Bedürfnisse in der Regel mit zusätzlichen Build Agents abgedeckt werden.

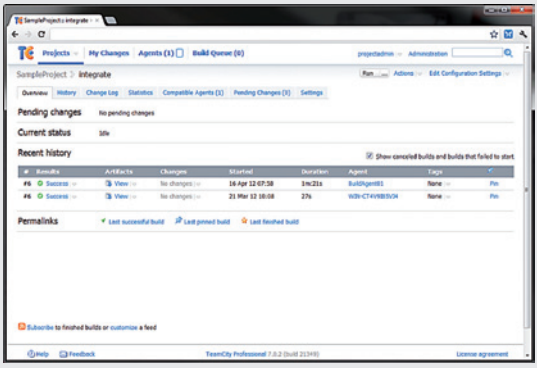
Stossen Sie mit den Ressourcen des Host-Systems an die Grenzen, gibt es zwei Möglichkeiten:

- Eine zusätzliche Host-Maschine (analog der ersten)
- Einen leistungsfähigeren Host mit mehreren Prozessoren, mehr RAM und einem RAID. Damit verbunden wäre auch ein Upgrade der VMWare vSphere Hypervisor auf eine kostenpflichtige Edition, da die Hypervisor Edition nur Hosts mit einem Prozessor erlaubt.

4.3 EINSTEIGERKONFIGURATION

Da der Einstieg in Continuous Integration selbst mit der Basiskonfiguration einen nicht zu unterschätzenden Aufwand darstellt, bietet

Abbildung 3:
Screenshot Beispielprojekt
im Web-Interface



bbv Software Services AG in Zusammenarbeit mit der Tochtergesellschaft bbv ICT Solutions AG eine Continuous-Integration-Umgebung als Dienstleistung an[10]. Bei dieser Dienstleistung erhalten Sie eine einsatzbereite Continuous-Integration-Umgebung mit einem konfi-gurierten Continuous-Integration-Server und einem Build Agent in einer virtualisierten Umgebung. Sie müssen lediglich Ihr VCS über eine Konfiguration einbinden, damit der Quellcode verfügbar ist, und ihren Build Prozess abbilden. Um den Einstieg zu erleichtern, ist auf dem System bereits ein lauffähiges Beispielprojekt eingerichtet.

[10] http://www.bbv-ict.ch/

5 FAZIT

Eine Continuous-Integration-Umgebung aufzubauen und Projekte darin zu hosten, ist eine herausfordernde Aufgabe, für die auch einiges an Erfahrung notwendig ist. Effizient angepackt, wird die Aufgabe in kleine Schritte unterteilt und die Continuous-Integration-Umgebung fortlaufend in ihren Möglichkeiten erweitert. Abhängig davon, wie viele der Voraussetzungen aus Kapitel 3 und welche Zwischenziele ebenfalls aus Kapitel 3 bereits vollständig oder teilweise erfüllt sind, ist die Umsetzung mehr oder weniger aufwendig.

Mit Continuous Integration können Integrationskosten reduziert werden, weil die Integration laufend durchgeführt wird, dadurch werden Fehler früh gefunden und sind günstiger in der Behebung.

Unsere Erfahrungen zeigen, dass Continuous Integration für Softwareprojekte, die eine hohe Qualität aufweisen wollen, unerlässlich ist. Für die Umsetzung von agilen Softwareprojekten ist Continuous Integration eine Voraussetzung.

Wenn Sie Continuous Integration richtig einsetzen, können Sie die Qualität Ihrer Softwareprojekte erhöhen und die Kosten senken. Continuous Integration ist dabei aber nur eines der Werkzeuge, das in erfolgreichen Projekten eingesetzt werden sollte. Nehmen Sie Kontakt mit uns auf, wenn auch Sie Continuous Integration und agile Softwareentwicklung erfolgbringend verwenden wollen.

6 ANHANG

6.1 AUTOR

Philipp Dolder

ist Spezialist für Continuous Integration bei bbv Software Services AG. Als Software-Architekt und Entwickler setzt er Continuous Integration seit mehreren Jahren erfolgreich in agilen Projekten ein. Dabei strebt er eine stetige Verbesserung in der Umsetzung des Prozesses an. Philipp Dolder ist Dipl. Inf. Ing. FH, Certified Scrum Master (CSM) und Microsoft Certified Professional Developer (MCPD).

6.2 QUELLENVERZEICHNIS

Beck, K. (1999). Extreme Programming Explained. Addison-Wesley.

Fowler, M. (1. Mai 2006). Abgerufen am 10. November 2008 von Continuous Integration:
http://www.martinfowler.com/articles/continuousIntegration.html

Holmes, M. (2005). Expert .NET Delivery Using NAnt and CruiseControl.NET. Apress.

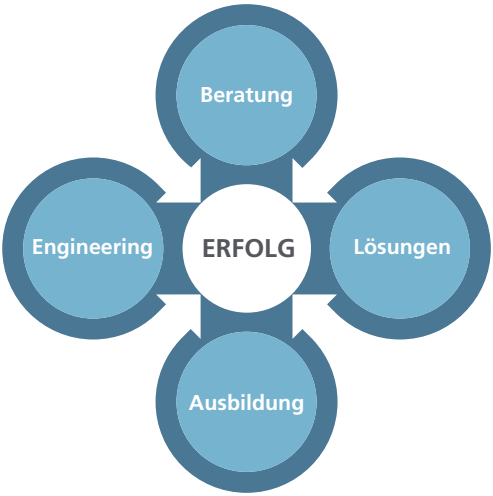
JetBrains. (2012). Teamcity Documentation. Abgerufen am 29. November 2012 von
http://confluence.jetbrains.net/display/TCD7/TeamCity+Documentation

7 GLOSSAR

Begriff	Erklärung
CI	Continuous Integration
CIE	Continuous Integration Environment. Die Umgebung (Hardware und Software), auf welcher der Integrationsprozess durchgeführt wird.
VCS	Version Control System. System zur versionierten Verwaltung von Quellcode
XP	Extreme Programming. Eine von Kent Beck entwickelte, agile Entwicklungsmethode
TDD	Test Driven Development

bbv
MAKING VISIONS WORK.

bbv Software Services AG ist ein Schweizer Software- und Beratungsunternehmen, das Kunden bei der Realisierung ihrer Visionen und Projekte unterstützt. Wir entwickeln individuelle Softwarelösungen und begleiten Kunden mit fundierter Beratung, erstklassigem Software Engineering und langjähriger Branchenerfahrung auf dem Weg zur erfolgreichen Lösung.



Unsere Booklets und vieles mehr finden Sie unter
www.bbv.ch/publikationen

MAKING VISIONS WORK.

www.bbv.ch · info@bbv.ch