Consulting Guild AG
Methodenberatung für Projekte im 21. Jahrhundert

Permanente Integration

Einstellung und Prozess versus Werkzeuge

Inhalt:

Version: 1.0

Status: **Draft** - aber immerhin schon mit dem Happy-Day Szenario

Nächste Schritte:

Task	Value	Cost
~~Happy-Day Szenario detaillieren~~	~~10~~	~~5~~
~~Happy-Day Scenario visualisieren~~	~~3~~	~~3~~
Story Schreiben (SUCCESS-Prinzip)	7	6
Begriffe beschreiben	7	2
Szenario mit Problemen detaillieren	7	3
Worst Case Szenario detaillieren	5	5
Weitere Szenarien visualisieren	5	12

Einleitung

In letzter Zeit erzähle ich immer wieder etwas über meine Sicht auf das Thema "Continuous Integration" – es wird Zeit, das mal von Flipcharts und Whiteboards in Bits und Bytes zu transponieren.

Permanente Integration ist aus meiner Sicht etwas ganz anderes als der reine Tooleinsatz. Auch wenn mittlerweile selbst im Wikipedia-Artikel über “Kontinuierliche Integration” das Hauptaugenmerk auf der Automatisierung von Build, Test und Deployment liegt, gibt es noch eine andere Art, sich dem Thema zu nähern.

Worum geht's hier überhaupt?

Permanente Integration (Continuous Integration) ist eine Techniken in der Softwareentwicklung, durch die es Endbenutzern bereits sehr früh – und vor allem zuverlässig – möglich ist, frisch entwickelte Software zu nutzen. Durch diese Zielsetzung – und weil sich auch für das Entwicklungsteam erhebliche Vorteile bieten – ist der Begriff "Continuous Integration" mittlerweile nahezu etabliert. Auf jeden Fall gehört es zum guten Ton in einem neuen Projekt auch das Thema der permanenten Integration zu adressieren.

Was aber genau bedeutet "permanente Integration" nun eigentlich genau?

Überblick

In einem Satz zusammengefasst: Permanente Integration ist eine Arbeitsweise, bei der der "Hauptzweig" (Mainline) der entwickelten Software nahezu permanent ein nutzbarer, lauf- und auslieferungsfähiger Stand des zu entwickelnden Produktes enthält und jede Entwickleraufgabe (Task) sofort nach ihrer Fertigstellung in diesen Hauptzweig integriert wird.

Um das zu erreichen ist kaum ein Einsatz hochspezialisierter Werkzeuge notwendig - ein diszipliniertes Vorgehen reicht völlig aus. Nun sind in diesem Satz allerdings ein paar Begriffe, die mehr umfassen, als auf den ersten Blick sichtbar ist. Das ist zum einen der Begriff Hauptzweig hinter dem sich das SCM-Muster "Mainline"[1] verbirgt, zum anderen der Bergriff Entwickleraufgabe als verfeinerte Übersetzung von "Task". Auf diese beiden Begriffe komme ich gleich zurück, vorher möchte ich aber noch das Wort "integriert" etwas genauer betrachten. Im Umfeld der permanenten Integration verstehe ich unter "integriert", dass alle Tests des gesamten Produktes fehlerfrei laufen, die neuen Änderungen über eigene Tests verfügen und zudem in einem Versionsverwaltungssystem hinterlegt sind.

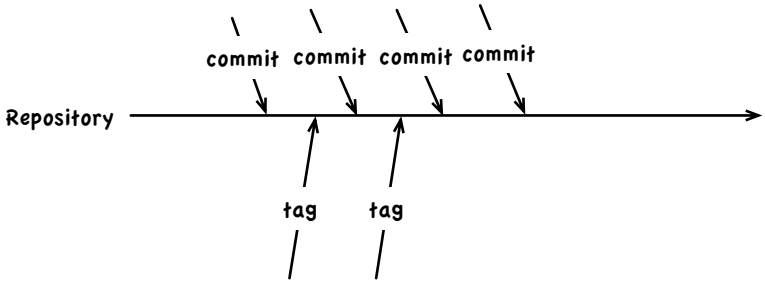
Permanente Integration ohne CI-Server

Auch ohne spezialisierte Tools - obwohl ein dedizierter Rechner für die Integration sinnvoll ist - lässt sich permanente Integration umsetzen.

Ich komme dazu noch mal auf das Konzept der Mainline – also der Idee, dass es einen stabielen Hauptzweig der Entwicklung gibt – zurück. Bei diesem Konzept gehen wir über das reine Versionieren von Arbeitsständen hinaus und bilden einzelne Arbeitszweige, die wir am Ende der Arbeit wieder mit dem Hauptzweig zusammenführen.

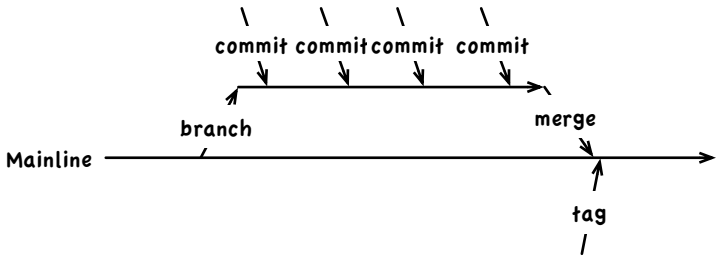
Zunächst ein kurzer graphischer Überblick über das konventionelle Modell aus Sicht des Repositories:

1 Sehr viel ausführlicher ist das Konzept der Mainline übrigens in [Berczuk02] erläutert – zusammen mit vielen weiteren sinnvollen Muster für die Versions- und Konfigurationsverwaltung deren Kenntnis ebenfalls in nahezu jedem Projekt weiterhelfen kann.



Unabhängig davon, woher die einzelnen Entwickler ihre Arbeitsversionen bekommen, wird im konventionellen Modell auf einem zentralen Repository gearbeitet und wichtige Änderungen an den gerade bearbeiteten Elementen des Systems werden vom Entwickler dorthin eingecheckt. Die wichtigste Verwaltungsgröße ist in diesen Umgebungen oft das "taggen" bestimmter Zustände um zu einem späteren Zeitpunkt genau diesen Stand wieder herstellen zu können.

Das Konzept der Mainline fordert ein anderes Vorgehen. Hier wird eben gerade nicht im zentralen Repository gearbeitet, vielmehr befindet sich dort immer nur ein "sauberer" Stand. Um dennoch ohne Angst vor Änderungen oder sogar vor dem Löschen von Inhalten arbeiten zu können, wird auf lokalen Zweigen gearbeitet, die ebenfalls unter Versionskontrolle stehen und am Ende der Arbeit wieder mit dem zentralen Repository zusammengeführt werden.



Je nach Werkzeug kann dabei das "branchen" und "mergen" des Hauptzweiges zwar durchaus aufwendig werden, aber selbst ganz ohne Versionsverwaltung ist dieses Konzept durchführbar, solange man Werkzeuge einsetzt, die eine Momentaufnahme (Snapshot) des Dateisystems ermöglichen.

Um vom Konzept der Mainline zum Konzept der permanenten Integration zu kommen muss allerdings noch mehr getan werden, wie die folgende Geschichte - in enger Anlehnung an Kent Beck [BeckXP] zeigt.

- Dieser kann sich - bei den heute vorhandenen Systemen – aber nur um die Automatisierung von Build, Test und Deployment kümmern.

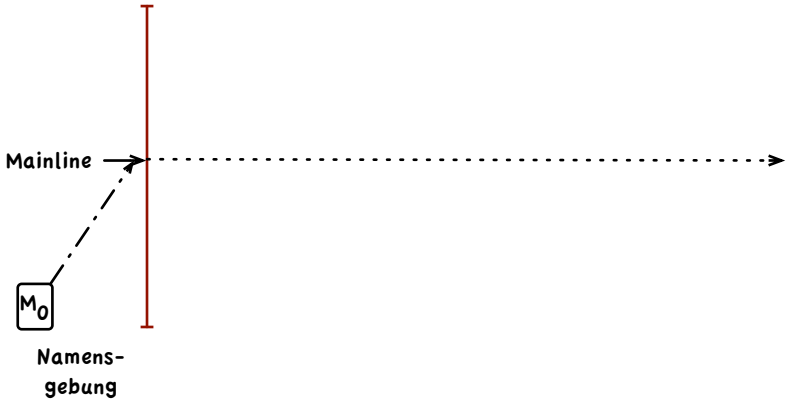
- Permanente Integration wurde allerdings ursprünglich (beispielsweise im 1. XP Buch von Kent Beck) als Vorgehensweise beschrieben und als solche sollte es meiner Ansicht nach auch heute noch gelebt werden.

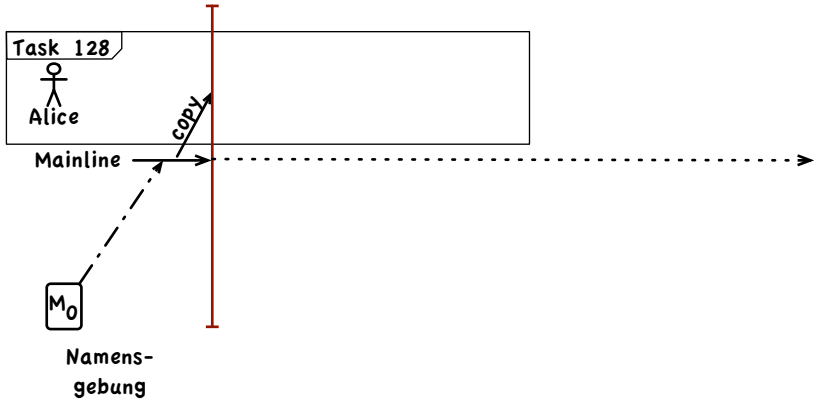
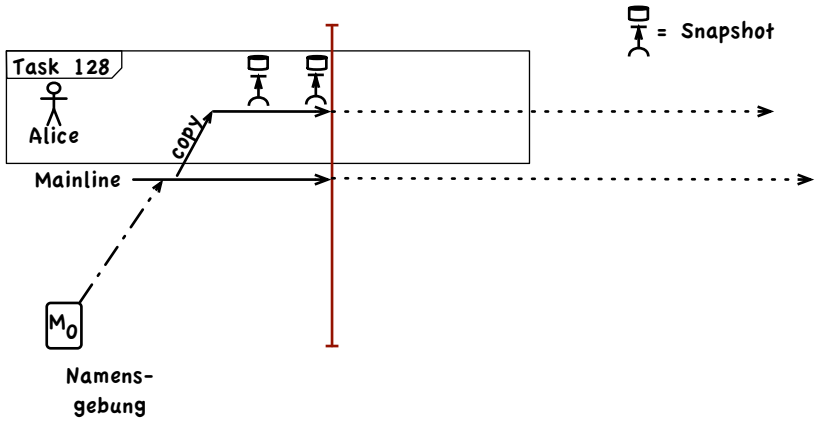
Für den Alles-geht-gut-Fall sind sehr einfache Szenarien für permanente Integration möglich, aber auch komplexere Fälle lassen sich nach dem gleichen Muster lösen.

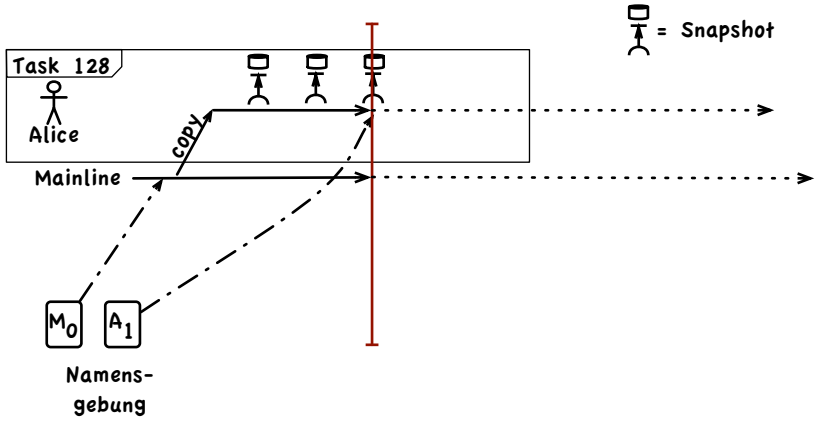
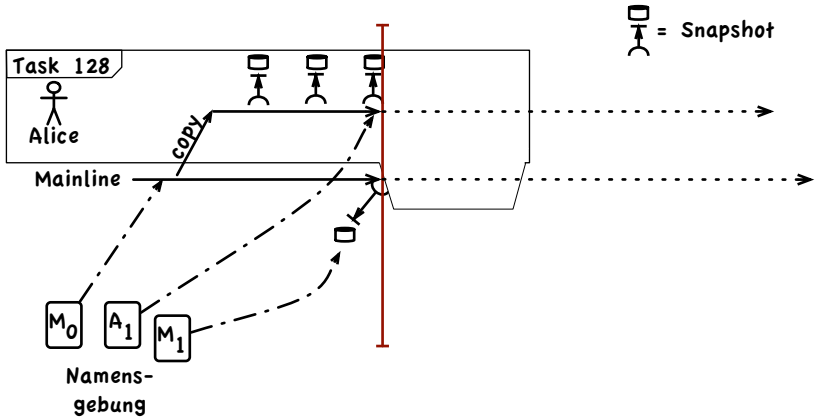
Wie sieht Continuous Integration aus?

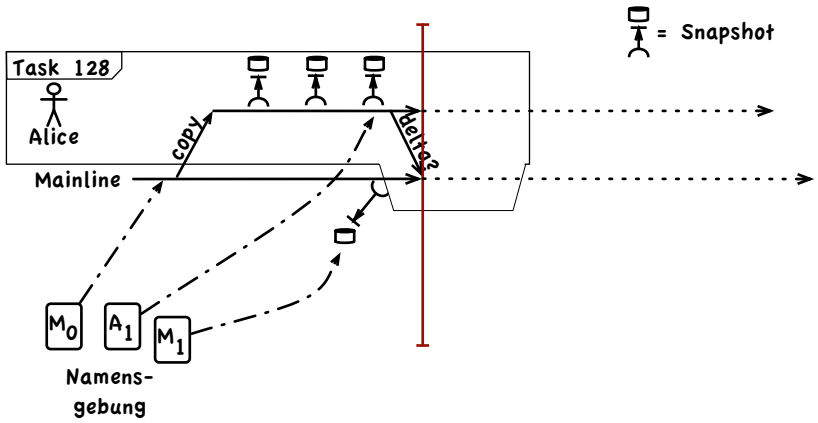
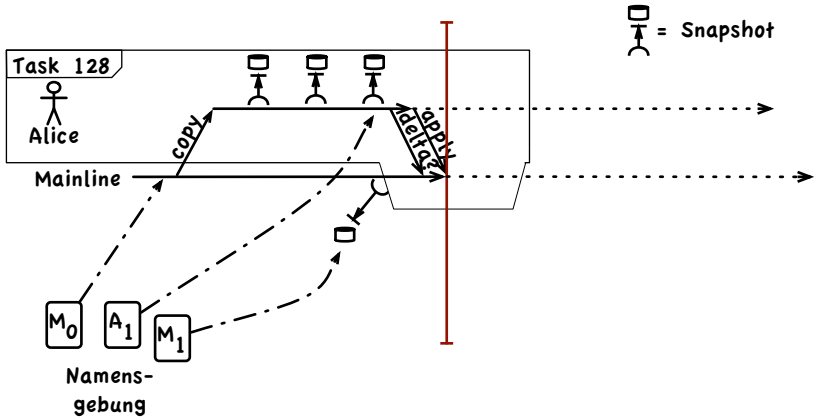
In diesem Überblick wird dargestellt, wie der Prozess an sich aussieht - eine Abbildung auf konkrete Umgebungen (Unix-Nativ, Git, Subversion, CVS) ist in Zusammenarbeit mit verschiedenen Freunden und Kollegen für die nächsten Wochen geplant.

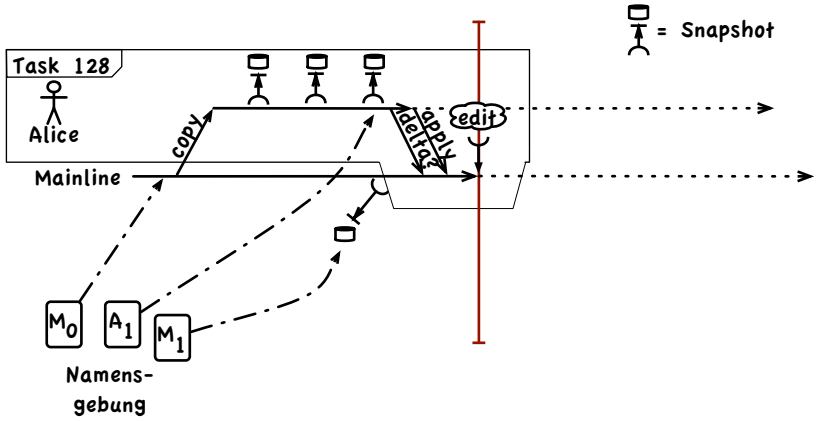
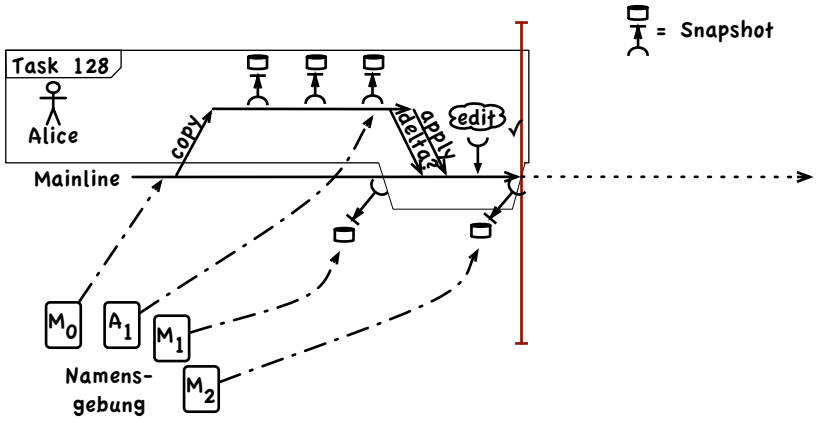
Continuous Integration im Happy-Day Szenario:

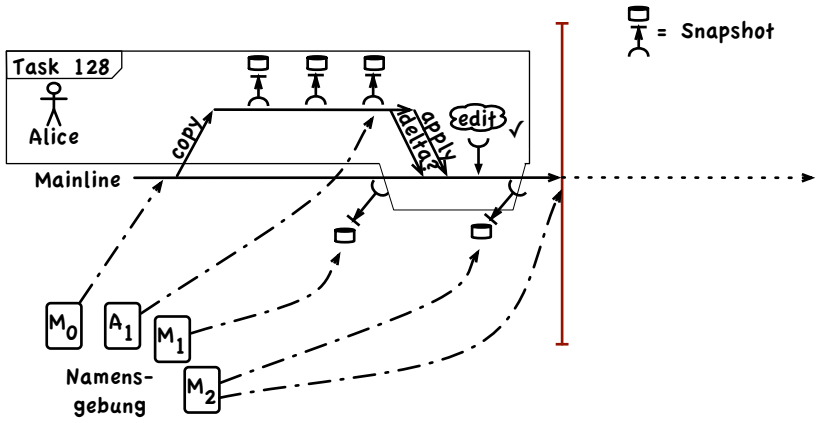
<table>
<tr><th>Nr.</th><th colspan="2">Beschreibung</th></tr>
<tr><td rowspan="2">1</td><td colspan="2"> 
</td></tr>
<tr><td>Zustand</td><td>In der Integrationsumgebung ist eine nominell fehlerfreie, lauffähige Instanz des zu entwickelnden Systems (einschliesslich der dazugehörenden Quellen) installiert und ein Abbild davon befindet sich als Mainline in der Quellcodeverwaltung. Nennen wir sie M0. (Eine leere Zielumgebung komplett ohne Sourcen erfüllt diese Bedingung auch!)</td></tr>
</table>

Nr.	Beschreibung	
2		
	Aktion	Der Entwickler erzeugt einen *Branch* von der aktuellen *Integrationsumgebung*.
	Zustand	In der *Integrationsumgebung* ist immer noch die lauffähige Version M0 verfügbar. Der Entwickler hat einen *lokalen Branch* der *Mainline* der ebenfalls lauffähig sein sollte.
3		
	Aktion	Der Entwickler beginnt mit der Arbeit an seinem *Task* und modifiziert seinen *Quellcode*. Zu von ihm gewählten Zeitpunkten erzeugt er *Sync-Points* zu denen er im Fehlerfall zurückkehren kann.
	Zustand	In der *Integrationsumgebung* ist immer noch die lauffähige Version M0 verfügbar. Der Entwickler hat einen *lokalen Branch* der *Mainline* mit mehreren (eventuell benannten) Zwischenständen.

Nr.	Beschreibung	
4		
	Aktion	Der Entwickler schliesst seinen *Task* ab und schreibt zu diesem Stand in seinem Branch einen benannten *Sync-Point*.
	Zustand	In der *Integrationsumgebung* ist immer noch die lauffähige Version M0 verfügbar. Der Entwickler hat einen *lokalen Branch* mit mehreren (eventuell benannten) Zwischenständen und einem getesteten und benannten Abschlussstand zu seinem *Task*.
5		
	Aktion	Der Entwickler akquiriert ein Lock auf den Integrationsprozess (zum Beispiel einfach indem er sich an die Integrationsmaschine setzt) $$$ Warum noch ein Syncpoint??? $$ und erstellt dabei einen Sync-Point für die Mainline.
	Zustand	In der *Integrationsumgebung* ist immer noch die lauffähige Version M0 verfügbar. Der Entwickler hat einen *lokalen Branch* der *Mainline* mit mehreren (eventuell benannten) Zwischenständen und einem getesteten Abschlusstand zu seinem Task. Die Integrationsumgebung ist für alle anderen Entwickler gesperrt.

<table>
<tr><th>Nr.</th><th colspan="2">Beschreibung</th></tr>
<tr><td rowspan="2">6</td><td colspan="2"> 
</td></tr>
<tr><td>Aktion</td><td>Der Entwickler ermittelt die Unterschiede zwischen seinem Branch und der Mainline.</td></tr>
<tr><td rowspan="3">7</td><td colspan="2"> 
</td></tr>
<tr><td>Aktion</td><td>Sofern es keine Konflikte gibt, wenn also in der Mainline keine Änderungen an Elementen vorgenommen wurden, die auch im Branch des Entwicklers geändert wurden, können die Änderungen, die der Entwickler durchgeführt hat, in die Mainline übernommen werden.</td></tr>
<tr><td>Zustand:</td><td>Der Entwickler hat einen lokalen Branch mit mehreren (eventuell benannten) Zwischenständen und einem getesteten Abschlusstand zu seinem Task. Die Integrationsumgebung ist für alle anderen Entwickler gesperrt. In der Integrationsumgebung ist ein nicht getesteter Stand der Kombination von Mainline und Änderungen durch den Task des Entwicklers.</td></tr>
</table>

<table>
<tr><th>Nr.</th><th colspan="2">Beschreibung</th></tr>
<tr><td rowspan="3">8</td><td colspan="2"> 
</td></tr>
<tr><td>Aktion</td><td>Der Entwickler versucht eine definierte Zeit lang (z.B. 30 Minuten) die Integrationstests (einschliesslich seiner neuen Tests) zum Laufen zu bekommen und ist dabei erfolgreich. (Anmerkung: Während der Integration ist es ihm erlaubt, auch Änderungen an anderen Teilen der Software vorzunehmen solange die Tests selber davon unberührt bleiben.)</td></tr>
<tr><td>Zustand</td><td>Der Entwickler hat einen lokalen Branch mit mehreren (eventuell benannten) Zwischenständen und einem getesteten Abschlusstand zu seinem Task. In der Mainline und der Integrationsumgebung befindet sind eine lauffähige und getestete Version des Gesamtsystems. Die Integrationsumgebung ist für andere Entwickler gesperrt. (Anmerkung: In einem realen Szenario ist es an dieser Stelle durchaus denkbar, dass der Entwickler-Branch und die Mainline nicht deckungsgleich sind.)</td></tr>
<tr><td rowspan="2">9</td><td colspan="2"> 
</td></tr>
<tr><td>Aktion</td><td>Der Entwickler erzeugt einen Sync-point des aktuellen Standes der Mainline, nennen wir ihn M2. Er eliminiert seinen Branch und hat damit den Task abgeschlossen.</td></tr>
</table>

Nr.	Beschreibung	
10	Zustand	In der *Integrationsumgebung* ist eine nominell fehlerfreie, lauffähige Instanz des zu entwickelnden Systems (einschliesslich der dazugehörenden *Quellen*) installiert und ein Abbild davon befindet sich als *Mainline* in der *Quellcodeverwaltung*. Im Grunde herrscht also wieder der gleiche Zustand, wie zu Beginn der Bearbeitung, nur mit einem neueren Stand und einer einer Anwendung, die mehr Funktionalität umfasst.
		

Anmerkung: Praktiken wie *Behaviour Driven Design*, *Test First* und *Pair-Programming* sind orthogonal zum Thema Continuous Integration und können sowohl zusätzlich zu als auch unabhängig von Continuous Integration eingesetzt werden.

Quellen:

[BeckXP] Die erste Auflage des Buches XP explained

[FowlerCI] Der Originalartikel von Fowler zum Thema CI

[Berczuk02] Software Configuration Management Patterns: Effective Teamwork and Practical Integration (Software Patterns Series) (Taschenbuch)