Wissenschaftlicher Fachtext

Software Quality Management (QM-VU)
Teil 2

Praktiken als Voraussetzung zur Verkürzung von Releasezyklen qualitativ hochwertiger Software

Name	Matr-Nr. / Kennz.	E-Mail Adresse
Bernhard Fleck	0325551 / 937	bernhard.fleck@gmail.com
Claus Polanka	0225648 / 534	e0225648@student.tuwien.ac.at

Status: Abstract
Datum: 17. Dezember 2011

Inhaltsverzeichnis

Abstract

Diese Arbeit beschäftigt sich mit der Ausarbeitung von qualitätssichernden Maßnahmen, die es Software-Unternehmen ermöglichen sollen, Deploymentzyklen drastisch zu verkürzen um rasch auf Veränderungen am Markt reagieren zu können. IT-Unternehmen die nur einmal jährlich eine neue Version ihrer Produkte zur Verfügung stellen, könnten künftig große Probleme haben konkurrenzfähig zu bleiben. Diese Arbeit befasst sich daher mit der Frage, welche Entwicklungspraktiken zu einem bestimmten Vorgehensmodell hinzugefügt bzw. von diesem entfernt werden müssen um die Deploymentgeschwindigkeit von jährlichen auf dreimonatige, auf monatliche, auf wöchentliche, auf tägliche und zu guter Letzt auf stündliche Zyklen zu verkürzen. Da sich die Disziplin des Software-Engineerings nicht verallgemeinern lässt, können bestimmte Techniken für bestimmte Deploymentzyklen positive, für andere jedoch negative Auswirkungen haben. Wenn man sich vorstellen würde, dass man zwei Entwicklungsteams, die mit unterschiedlich langen Deploymentzyklen arbeiten, nach ihren Praktiken befragen würde, was würde man als Antwort bekommen? Eines ist klar, Software-Entwickler müssen prinzipiell dieselben Probleme lösen und zwar von der Idee bis zum tatsächlichen Bereitstellen des Produkts für den Endanwender. Nur die Art und Weise wie die Software umgesetzt wird unterscheidet sich dramatisch je nach Länge des verwendeten Deploymentzyklus. In dieser Arbeit wollen wir detailliert darauf eingehen, welche Techniken notwendig sind, um einerseits

hohe Qualität der Software zu garantieren und um andererseits die Entwicklungszyklen drastisch zu verkürzen.

1 Einleitung

Allgemeine Einleitung der Thematik. Vielleicht eine Abgrenzung zwischen Continuous Delivery und Continuous Deployment vornehmen.

2 Related Work

Konzepte von Entwicklung und Deployment beschreiben. Die Frage herausarbeiten warum kürzere Releasezyklen notwendig/besser sind. Dabei nicht auf die Qualitäsaspekte vergessen.

3 Research Questions (Best-Practices)

Beschreibt und beantwortet die in Related Work erarbeitete Fragestellung indem Best Practices vorgestellt/herausgearbeitet werden. Weiters wird unser Vorgehen beschrieben. Und zwar wie iterativ eine immer kleinere Kadenz erreicht werden kann. In den Unterkapiteln werden die einzelnen Iterationsschritte vorgestellt.

3.1 Jahr auf Quartal

3.2 Quartal auf Monat

3.3 Monat auf Woche

3.4 Woche auf Tag

4 Success Stories / Case Studies

Um dem Ganzen einen fundierten Unterbau zu geben ein paar Success Stories vorstellen, welche bereits CD erfolgreich einsetzen. Gut wäre es hier natürlich wenn es nicht nur Beispiele aus der Industrie sind, sondern wissenschaftliche Veröffentlichungen.

4.1 IMVU

Als erstes Beispiel dient IMVU [1], eine soziale Online Community, in der in einer virtuellen Realität mit Hilfe von 3D Avataren geplaudert, Spiele gespielt und eigene Inhalte erschaffen und ausgetauscht werden können.

IMVU war eines der ersten *Lean Startup* Unternehmen welche Continuous Deployment aktiv einsetzten. Dabei ist dies aber nicht vorab im Ganzen geplant worden, sondern inkrementell entstanden. Derzeit sind bei IMVU zirka 50 technische Mitarbeiter angestellt. Ein wichtiger Punkt warum bei vorallem so vielen Entwicklern Continuous Deployment funktioniert, ist, dass es ein zentraler Bestandteil der Firmenkultur ist.
Als Vorteile von Continuous Deployment werden von IMVU die folgenden Punkte genannt:

- Regression wird sehr rasch erkannt
- Fehler können schneller behoben werden, da zwischen dem einspielen eines Fehlers und der Meldung über ein Problem nicht viel Zeit vergeht
- Der Release einer neuen Version erzeugt keinen zusätzlichen Overhead
- Als Feedback bekommen sie sofort messbare Kerndaten von echten Kunden

Workflow

Eine wichtige Grundvoraussetzung für Continuous Deployment bei IMVU ist wie schon in Abschnitt 3.1 auf Seite 3 gezeigt: Continuous Integration. Als Technologie kommt hier Buildbot[2] zum Einsatz. Um die Vorteile von Continuous Integration voll ausnützen zu können wird beim Entwickeln selbst *Commit Early Commit Often* praktiziert. Ist ein Feature fertig entwickelt, oder eine Bug behoben worden, werden zuerst lokale Tests auf der Entwicklermaschine durchgeführt. Wenn all diese Tests positiv durchlaufen wurden, wird der neue Code in Quellcode Versionsverwaltung eingespielt.

Erst jetzt werden sämtliche Tests der Test-Suite angestoßen. Zur Zeit sind dies zirka 15.000 Tests aus den Bereichen Unit-Tests, Funktions-Tests und Verhaltens-Tests. Dabei werden die folgenden Technologien eingesetzt:

- Selenium Core wird mit einem eigens entwickelten API Wrapper für die Verhaltens-Tests eingesetzt
- YUI Test wird für Browser bsierte JavaScript Unit-Tests verwendet
- PHP SimpleTest
- Erlang EUnit
- Python UnitTests

Schalgt nur einer der Tests fehl wird der zuletzt eingespielte Code zurückgesetzt. Es ist zu beachten, dass nicht nur die Masse an Tests, sprich die Testabdeckung, wichtig für

[1] IMVU: `http://www.imvu.com/`
[2] Buildbot: `http://trac.buildbot.net/`

IMVU ist, sondern auch die Qualität der Tests. Ein weiteres Merkmal ihrer Firmenkultur ist das Schreiben von qualitativ hochwertigen Tests. Durch diese Maßnahmen, also dem Schreiben von gründlichen hochwertigen Tests, welche sich auf alle Aufgabenbereiche verteilen, schafft es IMVU ein separates Qualitätssicherungsteam überflüssig zu machen.

Nachdem sämtliche Tests erfolgreich durchgeführt wurden, wird ein eigen entwickeltes Build-Skript angestoßen um den neuen Code in die Produktionsumgebung einzuspielen. IMVUs Produktionsumgebung besteht aus einem Custer mit derzeit zirka 700 Servern. Das Build-Skript verteilt zwar den Code im gesamten Cluster, umgestellt werden zunächst aber nur eine geisse Prozent Anzahl an Servern. Die Umstellung auf den neuen Code erfolgt recht simpel per Symlink.

Durch ein ständig aktives Monitoring werden fortlaufend Messwerte über den Gesundheitszustand des Clusters gesammelt. Diese Messdaten beinhalten Werte für CUP-, Speicher- und Netzwerk-Last, aber auch Business Metriken kommen zum Einsatz. Findet nach einer gewissen Zeitspanne keine Regression des Clusters statt wird der neue Code auf allen Servern aktiv geschalten. Durch das begleitende Monitoring könnte so noch immer jederzeit auf die vorherige Version zurückgewechselt werden.

Kurz sei noch erwähnt wie bei IMVU mit den ralationalen Datenbanken verfahren wird. Da ein Datenbank Schema Rollback nur schwer möglich ist, bzw. das Verändern des Schemas einen schwerwiegenden Eingriff darstellt, durchlaufen Schemamodifikationen, im gegensatz zum Code Deployment, einen formalen Review Prozess. Müssen tatsächlich die Strukturen der Tabellen angepasst werden, bleiben die alten Tabellen weiterhin bestehen und es werden einfach neue Tabellen mit der neuen Struktur erstellt. Die Daten werden dann per *Copy on Read* bzw. per Hintergrund-Job migriert.

4.2 Huitale

4.3 Digg 4

4.4 WiredReach

4.5 Wealthfront

5 Schlussfolgerungen

Kurze Zusammenfassung der „Ergebnisse“, bzw. Auflisten der Vorteile wenn man sich an die genannten Best Practices hällt. Dabei kann man sich auch stark an dem Abstract orientieren.

Literatur

[1] Puneet Agarwal. „Continuous SCRUM : Agile Management of SAAS Products“. In: *Proceedings of the 4th India Software Engineering Conference*. 2011, S. 51–60. ISBN: 9781450305594.

[2] Jasper Boeg. *Priming Kanban*. Trifork, 2011, S. 81. ISBN: 0984521402. URL: http://www.infoq.com/resource/minibooks/priming-kanban-jesper-boeg/en/pdf/Primingkanban-JesperBoeg.pdf.

[3] T Chow und D Cao. „A survey study of critical success factors in agile software projects“. In: *Journal of Systems and Software* 81.6 (Juni 2008), S. 961–971. ISSN: 01641212. DOI: 10.1016/j.jss.2007.08.020.

[4] Timothy Fitz. *Continuous Deployment*. 2009. URL: http://timothyfitz.wordpress.com/2009/02/08/continuous-deployment/.

[5] Timothy Fitz. *Continuous Deployment at IMVU: Doing the impossible fifty times a day*. 2009. URL: http://timothyfitz.wordpress.com/2009/02/10/continuous-deployment-at-imvu-doing-the-impossible-fifty-times-a-day/.

[6] Jez Humble. *Continuous Delivery vs Continuous Deployment*. 2010. URL: http://continuousdelivery.com/2010/08/continuous-delivery-vs-continuous-deployment/.

[7] Jez Humble, Chris Read und Dan North. „The Deployment Production Line“. In: *Proceedings of the conference on AGILE 2006*. IEEE Computer Society, 2006, S. 113–118. DOI: 10.1109/AGILE.2006.53.

[8] Jez Humble und MoleskyJoanne. „Why Enterprises Must Adopt Devops to Enable Continuous Delivery“. In: *Cutter IT Journal* 24.8 (2011), S. 6–12.

[9] J Humble und D Farley. *Continuous Delivery: Reliable Software Releases Through Build, Test, and Deployment Automation*. Addison Wesley Signature Series. Addison-Wesley, 2010. ISBN: 9780321601919.

[10] Jingyue Li, Nils B Moe und Tore Dybå. „Transition from a Plan-Driven Process to Scrum – A Longitudinal Case Study on Software Quality“. In: *Science And Technology* (2010).

[11] Fergal Mccaffery u. a. „An Agile process model for product derivation in software product line engineering“. In: *Journal of Software Maintenance and Evolution: Research and Practice* (2010). DOI: 10.1002/smr.

[12] Subhas Chandra Misra, Vinod Kumar und Uma Kumar. „Identifying some important success factors in adopting agile software development practices“. In: *Journal of Systems and Software* 82.11 (Nov. 2009), S. 1869–1890. ISSN: 01641212. DOI: 10.1016/j.jss.2009.05.052.

[13] Marko Taipale. „Huitale - A Story of a Finnish Lean Startup“. In: *Lean Enterprise Software and Systems*. Springer Berlin Heidelberg, 2010, S. 111–114.

[14] Jiangping Wan. „Empirical Research on Critical Success Factors of Agile Software Process Improvement“. In: *Journal of Software Engineering and Applications* 03.12 (2010), S. 1131–1140. ISSN: 1945-3116. DOI: 10.4236/jsea.2010.312132.