Duale Hochschule Baden-Württemberg
Mannheim

Projektarbeit 1

Optimierung des Release-Prozesses bei SAP TwoGo mithilfe von Continuous Delivery

Studiengang Wirtschaftsinformatik Software Engineering

Bearbeitungszeitraum: 08.08.2013 bis 08.11.2013

Verfasser	Johannes Haaß
Matrikelnummer	4101368
Kurs	WISE12B
Studiengangsleiter	Prof. Dr. Thomas Holey
Ausbildungsfirma	SAP AG Dietmar-Hopp-Allee 16 Walldorf
Firmenbetreuer	Dirk Lehmann dirk.lehmann@sap.com
Wissenschaftlicher Betreuer	Tobias Kißmer tobias.kissmer@schaeffler.com

Abstract

Mitfahrgelegenheiten erleben momentan eine starke Nachfrage. Besonders kleine Unternehmen und Start-ups haben diesen Trend erkannt und entsprechende Cloud-Lösungen auf den Markt gebracht. Durch schlanke Strukturen und flache Hierarchien können diese sehr schnell neue Funktionen an den Kunden bringen. Große Unternehmen, wie beispielsweise SAP, können durch ihre Größe dieses Tempo nicht mitgehen und sind damit im Bereich der Cloud-Lösungen im Nachteil.

Im Rahmen dieser Projektarbeit werden Optimierungen durchgeführt, um diesen Nachteil auszugleichen. Dazu wird der Release-Prozess der SAP Mitfahrlösung TwoGo analysiert und den zentralen Merkmalen von Continuous Delivery gegenübergestellt. Aus dieser Gegenüberstellung werden potentielle Optimierungsmöglichkeiten für den Release-Prozess von SAP TwoGo abgeleitet. Diese betreffen sowohl technische als auch organisatorische Aspekte.

Eine konkrete Optimierungsmöglichkeit ist der Deployment-Test. Dieser überprüft den Deployment-Vorgang und erkennt dabei Fehler, Warnungen und Erfolge. Zur Umsetzung dieses Deployment-Tests wird der Deployment-Vorgang analysiert, ein Entwurf erstellt und dieser implementiert und getestet.

Inhaltsverzeichnis

Abkürzungsverzeichnis

CD Continuous Delivery

Abbildungsverzeichnis

1 Einleitung

1.1 Problemstellung und -abgrenzung

1.2 Ziel der Arbeit und Vorgehensweise

2 Projektmanagement

2.1 Anforderungen

Während der ersten Planung von Avalon wurde uns die Komplexität und der Aufwand dieses Projektes deutlich. Zur Unterstützung des Entwicklungsprozesses haben wir uns entschieden Programme einzusetzen, die Aufgaben des Projektmanagements abnehmen bzw. erleichtern. Für uns ergaben sich zwei zentrale Anforderungen an diese Programme. Die erste Anforderung ist es einen Gesamtüberblick über den aktuellen Stand des Projektes darzustellen. Dazu gehört, was noch erledigt werden muss, was gerade gemacht wird und was bereits erledigt wurde. Die zweite Anforderung ist eine zentrale Codeverwaltung mit Versionsverwaltung, um das gemeinsame und zeitgleiche Programmierung zu ermöglichen. Außerdem soll durch Versionsverwaltung gegen Datenverluste vorgebeugt werden.
Da wir oft von unterschiedlichen Orten und Rechnern arbeiten, müssen die genannten Anforderungen immer und überall erfüllt werden. Um diese, weitere Anforderung zu erfüllen lag es nahe, die Daten in der Cloud zu speichern.
Aufgrund von persönlichen Erfahrungen und Internet Recherchen haben wir entschieden, dass wir für den Gesamtüberblick *Trello* benutzen und für die Codeverwaltung *GitHub* benutzen. Beide Tools erfüllen die oben genannten Anforderungen.

2.2 Einrichtung von Trello und GitHub

Die Einrichtung von Trello ist sehr einfach, da keine Software installiert werden muss und alles im Browser läuft. Zur Einrichtung ist nur eine Registrierung und die Erstellung eines gemeinsamen *Boards* notwendig. Ein solches Board besteht aus beliebig vielen Spalten, in denen die konkreten Aufgaben stehen. Unser Avalon Board hat folgenden Aufbau:

To-Do	**Doing**	**Done**
Kaufalg.	Use-Cases	Marktanalyse
...	...	...

Auch die Einrichtung von GitHub ist recht einfach. Auch hier müssen sich die Mitglieder auf der GitHub Homepage registrieren. Danach erstellt ein Mitglied ein neues Repository und fügt die anderen Mitarbeiter als Collaborators hinzu. Zur Nutzung ist es noch notwendig *Git* oder die *GitHub-GUI* auf den einzelnen Rechnern zu installieren. Dieses Tool benötigt man um das Repository herunterzuladen und Änderungen hochzuladen. Da in unserem Team nur geringe Erfahrungen mit Git vorhanden waren, entschieden wir uns die GitHub-GUI zu nutzen.

2.3 Praktische Erfahrungen mit Trello und GitHub

Während Entwicklung von Avalon stellte sich besonders Trello als sehr hilfreiches, einfaches und fehlerfreies Tool dar. Die Funktionsweise wurde jedem sofort verständlich und war selbst erklärend. Mit diesen Eigenschaften konnte Trello immer einen guten Überblick über den aktuellen Stand liefern und erfüllte die von uns gestellten Anforderungen sehr gut.
GitHub dagegen lief nicht ganz so unproblematisch. Die Funktionsweise über die GUI war zwar sehr logisch und einleuchtend, allerdings kam es regelmäßig zu teils merkwürdigen Fehlern. Das größte Problem entstand, wenn zwei Mitglieder die gleiche Datei bearbeiteten. Das Mergen der Datei über die GUI war nie möglich und manuelles Mergen über die Konsole funktionierte nur selten. Um dieses Problem schnell und einfach zu Lösen, hat einer der beiden seine Änderungen separat gespeichert und das Repositroy neu heruntergeladen. Diese Fehler führte oft zu Frustrationen. Der Grund für diese Fehler lag sowohl an unsere Unerfahrenheit mit Codeverwaltung und an der GitHub GUI. Ein weiterer frustrierender Fehler war das Überschreiben von einigen Dateien mit älterem Inhalt. All diese Probleme führten zu Überlegungen GitHub durch ein anderes Tool, wie Dropbox, zu ersetzen. Diese Überlegungen wurden jedoch wieder verworfen, da mit Dropbox vermutlich noch weitere Probleme entstanden wären. Im Nachhinein wäre es wahrscheinlich besser gewesen anstatt der GUI die Git-Bash zu verwenden.

3 Anforderungsspezifikation

3.1 Zielbestimmung

Der Spieler soll ein Unternehmen steuern, dass Smartphones produziert und an Endkunden verkauft.

3.1.1 Muss-Kriterien

- Steuerung der Abteilungen: Einkauf, Produktion, Verkauf, Forschung, Marketing und Rechtsabteilung
- Spieler spielen gegen andere menschliche Spieler.
- Die Nachfrage nach Smartphones soll nach realistischen Vorgaben simuliert werden.
- Das Spiel läuft rundenbasiert ab. Eine Runde entspricht einer Periode.
- Spieler können über ein Netzwerk spielen, kein Hot-Seat!
- Abteilungen und Produkte können durch ein "Level up"verbessert werden.
- Zufallsereignisse sollen den Spielverlauf beeinflussen.

3.1.2 Kann-Kriterien

- Spiele Lobby mit Chat
- Downgrade von Abteilungen um Kosten einzusparen.

3.1.3 Abgrenzungskriterien

Aufgabe des Spieles ist es nicht sämtliche Aspekte eines Unternehmens zu simulieren. Es genügt wenn wesentliche Merkmale eines Smartphones-Herstellers abgebildet werden.

3.2 Einsatz

Die Zielgruppe dieses Unternehmensplanspiel sind Studenten und anderen Personen, die Interesse haben ein Unternehmen zu steuern, das Smartphones herstellt. Der Anwendungsbeireich spielt keine Rolle. Da die Software regelmäßig Eingaben benötigt, ist ein unbeaufsichtigter Betrieb nicht vorgesehen.

3.3 Umgebung

Die Software soll auf Windows-Rechnern laufen, auf denen einen aktuelle Java Version installiert ist. Die Hardware muss den Java Mindestanforderungen entsprechen und es muss eine Netzwerkkarte vorhanden sein.

3.4 Funktionalität

Typische Abläufe im Planspiel sind:

- Das Einkaufen von Rohstoffen.
- Das Produzieren von Smartphones.
- Das Verkaufen von Smartphones.
- Das Upgraden von Abteilungen.
- Das Starten von Forschungen und Spionagen.
- Das Starten von Marketingkampagnen.
- Das Überprüfen und Verklagen von anderen Spieler.

- Das Anfechten eines Rechtsstreites.

3.5 Daten

Benutzereingaben müssen nicht dauerhaft gespeichert werden. Die Startparameter für das Spiel sollen in einer Datei gespeichert werden.

3.6 Leistungen

Das Spiel soll zügig auf Eingaben reagieren. Wartezeiten sollen unter 10 Sekunden liegen.

3.7 Benutzeroberfläche

Die Benutzeroberfläche soll alle veränderbaren und nicht veränderbaren Parameter des Unternehmens darstellen. Zudem soll die Oberfläche minimalistisch und übersichtlich sein.

3.8 Qualitätsziele

Fehlerfreies Spielen des Spiels und gute Benutzbarkeit.

3.9 Ergänzungen

4 Praktischer Teil: Deployment-Test

5 Zusammenfassung und Ausblick

Literaturverzeichnis

Fowler, M. (2006), 'Continuous integration', `http://www.martinfowler.com/articles/continuousIntegration.html`. Abgerufen am 18.10.2013.

Hicks, J. (2012), 'Germany's carpooling.com proves rideshare works', `http://www.forbes.com/sites/jenniferhicks/2012/06/08/germanys-carpooling-com-proves-rideshare-works/`. Abgerufen am 18.10.2013.

Humble, J. und Farley, D. (2011), *Continuous delivery: reliable software releases through build, test, and deployment automation*, Addison-Wesley, Upper Saddle River, NJ [u.a.].

Kniberg, H. und Skarin, M. (2010), *Kanban and Scrum - Making the Most of Both*, Lulu Enterprises Incorporated.

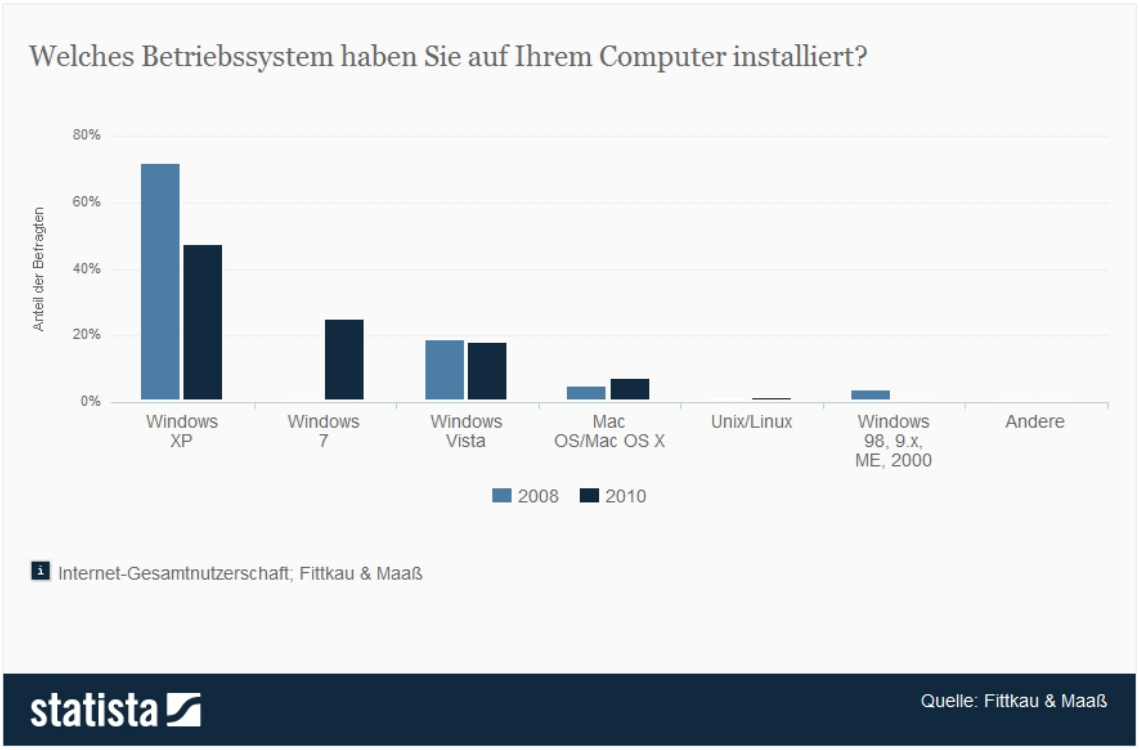
SAP AG (2013), 'SAP Geschäftsbericht 2012', `http://www.sap.com/corporate-de/investors/pdf/SAP-2012-Geschaeftsbericht.pdf`. Abgerufen am 18.10.2013.

Swartout, P. (2012), *Continuous Delivery and DevOps a Quickstart guide.*, Packt Pub., Birmingham.

Wiest, S. G. (2011), *Continuous Integration mit Hudson: Grundlagen und Praxiswissen für Einsteiger und Umsteiger*, dpunkt-Verl., Heidelberg.

Anhang

Abschnitt A Umfrage:



Abschnitt B Ausschnitt aus internem Wiki:

Project Information

TwoGo is a project about ride sharing and is intended for SAP employees only in the first phase. Later it shall be opened to everybody. It's sponsored by the sustainability group at SAP and focuses on our CO2 reduction goals.

The project hosts an outside-facing wiki, but relevant technical or team-internal information is stored in this team-facing wiki. Tickets are managed here as well. We used to use JIRA, switched now to Trac for its better cross-integration with source code repositories, build server and more.

Table of Contents

Development Infrastructure

The project uses Trac as project management tool with its Wiki for internal documentation, its Issue Tracker for stories, defects features and tasks, Perforce and git as VCS, JDK of version 6, Maven as build tool, Nexus to host your build artifacts and the build dependencies you require, Jenkins for continuous build and test, and Eclipse as IDE (see CodingStandard) for JAVA development, respectively WebStorm for JavaScript development. See the above links for detailed installation instructions. Developers just need to install a JDK, Perforce, Maven, and Eclipse with plugins plus the local server environment (identical with our central build and dev system). It includes the application and database server. Developers will need to install it so they can execute tests, run a full Maven build (which in turn executes the tests), run the application from Eclipse, or control the application server or inspect the database content.
In order to work our Android client or our new UI/landing page (a.k.a ScriptStorm) please see additional here

Another alternative is to use a Linux VMWare Image which is prepared for the TwoGo developer. The VMWare Image contains configured Perforce Client, Eclipse IDE, Lean Java Server and MaxDB. Please see VMImage for details.

Testing is treated as integral part of the development process and infrastructure. The TwoGo team runs unit, integration, scenario and specific DSL-based tests for all developed components. In addition Findbugs and Checkstyle are used for static code analysis (limited test scope to highest priority, but for them all issues must be cleared) and Cobertura for code coverage (50 % or higher must be reached for a stable Jenkins build and 80 % for a good "sunny" build). To ensure a reasonable quality and maintainability of the code please adhere to the Code Guidelines.

A demoable version can be accessed here https://spwdfvml0898.wdf.sap.corp:8443.

The latest JavaDoc (Backend only) can be accessed on our Jenkins build

Maven project information (Backend only) can be found here

Architecture and Concepts

There is a FeatureSpec available that explains some of the remaining features in more detail.

TwoGo was shifted away from River to a mainstream technology-mix platform with higher productivity and reliability. See here for more information.

TwoGo will facilitate the services of Nokia Maps (former Navteq). Details can be found here.

TwoGos web client is implemented in SAP UI5.

Release Management

TwoGo Codelines

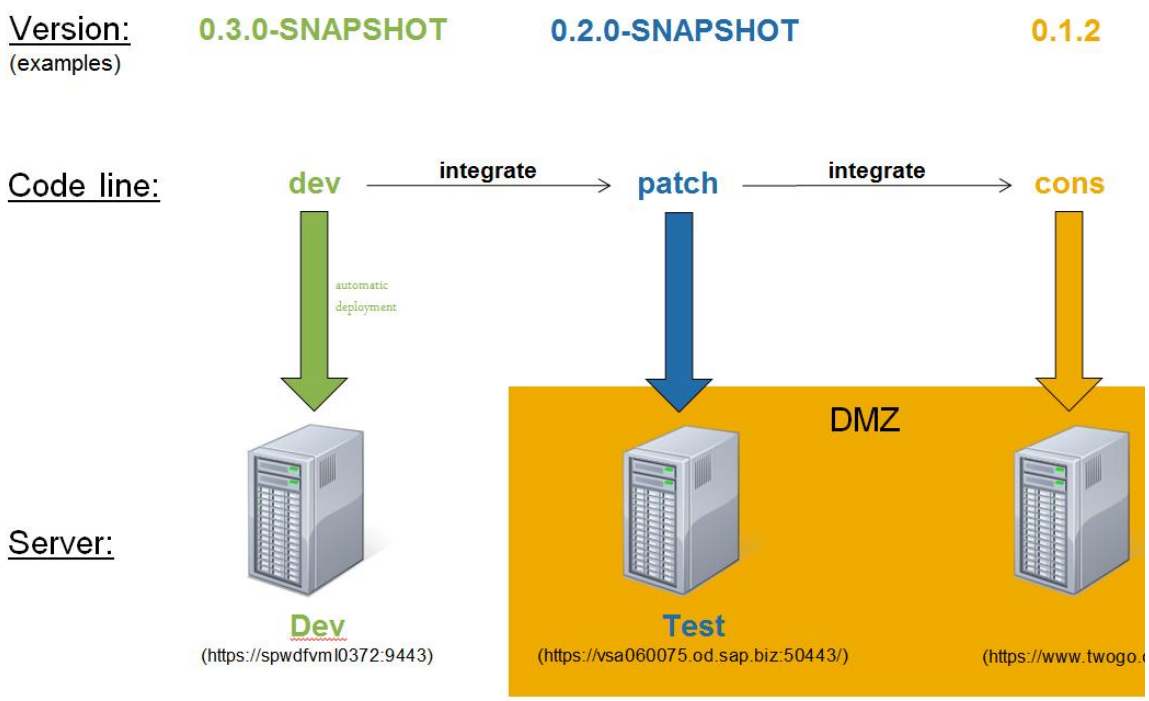
TwoGo source code is located in Perforce Port **perforce3406:3406**, depot **ptg.** The depot contains 4 (standard SAP) code lines: **dev, export, patch** and **cons**, whereas TwoGo only uses 3 of them: **dev, patch** and **cons** (and to be precise, TwoGo uses them not in their intended SAP functions).

- **dev** code line is owned by all TwoGo developers. No restriction applies to any check-ins, except the code must be buildable on CI Server (Jenkins: http://spwdfvm1246.wdf.sap.corp:8080/jenkins/). After each check-in an automatic deployment (incl. new change list) is triggered from CI Server to central Dev machine.
- **patch** code line is owned by TwoGo Quality Management (QM) Team. This code line is created by integrating dev code line to patch code line at a certain point in time (currently: every second week). TwoGo developers have full access (read/write) to this code line, however they're only allowed to change file(s) in patch code line on QM Team request. Deployments from patch code line will take place on request to central Test server.
- **cons** code line is owned by TwoGo Administrators. This code line is created by integrating patch code line to cons code line at a certain point in time (currently: every second week before dev gets integrated to patch). TwoGo developers have full access (read/write) to this code line, however they're only allowed to change file(s) in cons code line on Product Owner request (Hot Fix). Deployments from cons code line will take place on request to central TwoGo productive server (https://www.twogo.com).

TwoGo Versioning

TwoGo version numbers are formatted as proposed by Apache Maven project
`<major version>.<minor version>.<incremental version>-<qualifier>`

- TwoGo increases major version numbers to indicate incompatible API changes and/or new/reworked major features (incl. bug fixes)
- TwoGo increases minor version numbers to indicate compatible API changes and/or new/reworked features (incl. bug fixes)
- TwoGo increases incremental version number to indicate bug fix provisioning
- TwoGo uses Apache Maven qualifier `SNAPSHOT` to indicate instable (frequently changed) versions, which have not been officially released



1. After integrating patch code line to cons code line, the qualifier `SNAPSHOT` is removed from version number (in cons code line) to release (via TwoGo Nexus) the version.
2. In case a severe bug fix (a.k.a. HotFix) must be applied to productive server, the fix is checked in to cons code line, build on CI server and deployed to productive server. To identify this fix, TwoGo increments the incremental version number (e.g. `0.1.1` → `0.1.2`)
3. After integrating dev code line to patch code line, the minor version number gets incremented (in dev code line) (e.g. `0.2.0-SNAPSHOT` → `0.3.0-SNAPSHOT`)

In case TwoGo plans to release a new major version, the major version number gets incremented (e.g. `0.4.0-SNAPSHOT` → `1.0.0-SNAPTSHOT`). It is considered a best practice to reset all subsequent version numbers to 0, when incrementing a preceding version number. (e.g. `1.2.0-SNAPSHOT` → `2.0.0-SNAPHOT`) The qualifier `SNAPSHOT` remains, until patch code line gets integrated to cons code line.

Integrating code lines and incrementing TwoGo versions

Code line integration

Integrating code lines is done via SAP Perforce Management System.

Ehrenwörtliche Erklärung

Ich erkläre hiermit ehrenwörtlich:

1. dass ich meine Projektarbeit 1 mit dem Thema *Optimierung des Release-Prozesses bei SAP TwoGo mithilfe von Continuous Delivery* ohne fremde Hilfe angefertigt habe;
2. dass ich die Übernahme wörtlicher Zitate aus der Literatur sowie die Verwendung der Gedanken anderer Autoren an den entsprechenden Stellen innerhalb der Arbeit gekennzeichnet habe;
3. dass ich meine Projektarbeit 1 bei keiner anderen Prüfung vorgelegt habe;
4. dass die eingereichte elektronische Fassung exakt mit der eingereichten schriftlichen Fassung übereinstimmt.

Ich bin mir bewusst, dass eine falsche Erklärung rechtliche Folgen haben wird.

Mannheim, 11. November 2013

Johannes Haaß