Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Informatikdienste
ID Software Services
STC E 13
8092 Zürich

Dr. Benno Luthiger
Stampfenbachstrasse 67
+41 (44) 632 57 65
benno.luthiger@id.ethz.ch
www.foss.ethz.ch

Software Services (SWS)
Informatikdienste
ETH Zürich

Zürich, 30. Oktober 2013 BL

Jazoon 2013, Konferenz-Bericht

Joe Justice: *Test First Saves The World*

Wikispeed ist der Versuch, verschiedene im Software-Bereich eingeführte Methoden und Prozesse im Hardware-Bereich anzuwenden. Hardware meint in diesem Fall nicht Computer-Hardware, sondern ein Auto. Das gewünschte Auto soll sehr energiesparend sein, gleichzeitig auch unterhaltsarm und kostengünstig. Mit agilen Methoden (extreme manufacturing) und einer offenen und verteilt arbeitenden Community wurde bisher ein Prototyp entwickelt. Der Prototyp bestätigt die ursprünglichen Annahmen. Der Prototyp ist modular aufgebaut und besteht aus 8 Modulen. Als Antrieb kann sowohl ein Benzinmotor wie auch ein Elektromotor eingesetzt werden. In der aktuellen Version kostet der Prototyp $ 25‘000. Ein Exemplar des Prototyps kann in sehr kurzer Zeit erzeugt werden, wenn das Ausgangsmaterial vorhanden ist.

Diverse amerikanische Firmen (Boeing, Lockheed Martin) sind auf den Erfolg von Wikispeed aufmerksam geworden und haben den Wikispeed-Gründer Joe Justice eingeladen, mit ihren Ingenieuren Methoden-Workshops durchzuführen. In diesen Workshops bauten die Teams innerhalb einer Woche ein Exemplar des Wikispeed-Prototyps zusammen.

Konrad Malawski: *Continuous Delivery Antipatterns*

Das Problem bei Releasen ist, dass ein neuer Release neue Fehler einführen kann. Damit entsteht der Eindruck, dass Releasen riskant ist. Ein risiko-averses Projektmanagement könnte darauf drängen, möglichst selten Release zu machen. Dies führt aber zu einer Stagnation. Kleine Änderungen sollten schnell ausgebreitet werden.

Damit kontinuierliche Integration möglich wird, muss die Software eine Version haben. *Semantic Versioning* ist besser als ein beliebiger Hash-Tag.

Andres Almiray: *Spock: boldly go where no test has gone before*

Spock (http://code.google.com/p/spock/) ist ein Testing-Framework, welches für alle Sprachen eingesetzt werden kann, welche auf der JVM laufen. Spock ist Daten-getrieben und fällt dadurch auf, dass die Tests sehr einfach lesbar sind (setup, asserts etc.).

Dominique Guinard: *If Spock had an Android phone and the EVRYTHNG API*

Mit der Verbreitung von Smartphones haben sich auch Tags verbreitet, z.B. *QR-Codes*, *NFC* (near field communication) oder *EPC-RFID* (electronic product code). Als neue Perspektive taucht am Horizont *Bluetooth Low Energy* (BLE) auf. Für Android gibt es entsprechende Open-Source-Bibliotheken, um solche Tags zu lesen. Beispielsweise Zxing (http://code.google.com/p/zxing/, für QR-Code) oder der NFC-Leser, welcher seit Version 2.3.3 in der Android-Standard-Bibliothek enthalten ist.

Die im Code (QR oder NFC) gelieferte Information sollte eine URL sein, damit die effektive Information bei Bedarf aktualisiert werden kann.

Google hat mit dem Accessory Development Kit (ADK) ein Werkzeug geschaffen, mit welchem Arduino mit Android integriert werden kann. Auf dieser Plattform können Entwickler Applikationen erstellen, mit denen das Smartphone mit *embedded devices* kommuniziert.

Stefan Saasen: *True Git: The Great Migration*

Der Referent präsentierte seine Erfahrungen mit Git, welche er gemacht hat, seit Atlassian die Versionskontrolle auf diese Technologie umgestellt hat.

Git ist eine dezentrale Versionenverwaltung. Das bedeutet, dass das ganze Repository lokal verfügbar ist, mit allen Snapshots und der vollständigen History. Daraus ergeben sich für Git verglichen mit SVN klare Vorteile im Bereich Branching/Merging, weil Git von Anfang an auf solche Aktionen ausgerichtet war. Weil alle Informationen im lokale Repository enthalten sind, ist ein Branch in Git viel schneller erzeugt (und allenfalls wieder gelöscht) als bei SNV, wo alle notwendigen Informationen über das Netzwerk gehen. Zur Veranschaulichung der Schnelligkeit von Git präsentierte der Referent den Zeitaufwand für die Anzeige der Log-History eines Repositories: Git: 2 Sek., SVN: 9 Min.

Eine dezentrale Versionenverwaltung ermöglicht lokale Commits. Der Publikations-Prozess einer Änderung wird damit auf zwei Schritte aufgeteilt: Commt (lokal) und Publish (ins zentrale Repostory). Auf diese Weise werden ganz neue Prozesse möglich. Beispielsweise ist es viel einfacher, Code zu reviewen, weil der Review auf dem lokal eingebuchten Code erfolgen kann.

Atlassian hat die Git-Einführung in drei Phasen unterteilt: *inception* (Beginn), adaption (Anpassung), migration (Migration).
Beginn: Konversion testen; zu konvertierende Repositories identifizieren; Werkzeuge, welche angepasst werden müssen, identifizieren (IDE, Build-Werkzeuge etc.).
Anpassung: Entwickler müssen Prozesse und Einstellungen anpassen; Git-Enthusiasten identifizieren und als Helfer für andere Team-Mitglieder einsetzen; die Werkzeug-Unterstützung muss laufen; Training; Dokumentation.
Migration: siehe http://bit.ly/go-dvcs und http://www.slideshare.net/jazoon13/jazoon-2013stefansaasentruegitmigration?ref=http://guide.jazoon.com/

Pawel Wrzeszcz: *Visibility Shift in Distributed Teams*

Wenn die Team-Mitglieder am gleichen Ort arbeiten, so werden sie über ihre Präsenz sichtbar. Wenn die Team-Mitglieder verteilt arbeiten, so werden sie über ihre Leistung sichtbar.

Scrum ist auch bei verteilten Teams möglich. Scrum ist nicht dazu gedacht, allfällige Probleme zu lösen. Scrum macht solche Probleme aber sichtbar.

Vertrauen in die Leistungsbereitschaft der Team-Mitglieder (auch wenn kein Chef über den Rücken schaut) ist Voraussetzung dafür, dass verteilte Teams erfolgreich arbeiten.

Dierk König: *OpenDolphin: Enterprise Apps for Java Desktop, Web, and Mobile*

OpenDolphin soll die Entwicklung von Applikationen für mehrere Kanäle (Desktop, Android, iPhone, Embedded) ermöglichen. OpenDolphin realisiert diesen Anspruch, indem das Multi-Client-Konzept richtig und umfassen verwirklicht wird.
OpenDolphin setzt das grundlegende Prinzip der vollständigen Entkoppelung der Ansichten (Views) um. Änderungen, welche in einer View gemacht werden, werden immer über den Server synchronisiert. Das gelingt mit einem shared presentation model, d.h. einem Modell, welches von Server und Client geteilt und zwischen diesen synchronisiert wird.
Die konzeptionelle Trennung von Geschäftslogik und UI geht von der Annahme aus, dass die Geschäftslogik üblicherweise relativ stabil ist und auf dem Server ausgeführt wird, während das UI unter Umständen rasch verändert werden muss und deshalb möglichst schlank sein soll, damit es schnell ausgetauscht werden kann.

Für weitere Informationen siehe http://open-dolphin.org/dolphin_website/Home.html.

Sven Peters: *How To Do Kick-Ass Software Development*

Deliver Kick-Ass Software: Rasch ein Prototyp entwickeln (allenfalls ein Fake) und an Kunden testen. Möglichst grosse Anreize setzen für Kundenrückmeldungen, z.B. gut sichtbarer Feedback-Knopf mit einfachem Feedback-Formular auf jeder Seite der Web-Applikation.

One Kick-Ass Team: Einerseits sollen die Entwickler vor den Benützern geschützt werden, andererseits ist es auch für die Entwickler wichtig, dass sie die Arbeit des *Customer supports* und die Problem, die dort landen, kennen.
Allgemeiner Rat: Abteilungsgrenzen überwinden: Entwickler arbeiten für eine gewisse Zeit im Customer support und im Test-Team.

Kick-Ass Collaboration: Regeln zerstören einerseits den Arbeitsfluss, andererseits verhindern sie Fehler. Damit Regeln möglichst effektiv sind (um Qualität zu ermöglichen), sollen sie möglichst knapp sein: Branch – Pull (mit Code-Review) – Merge.
Für Kommunikation im Team sind Mails meist eine schlechte Lösung, Chat führt oft zu besseren Resultaten.

Kick-Ass Automation: Der Entwicklungsprozess soll möglichst weit automatisiert sein. Dabei soll viel Gewicht auf *fail fast* gelegt werden, damit die Feedback-Zyklen möglichst kurz sind.

Lorenz Meier: *Open Safety For Drones*

Der Referent ist Doktorand in der *Computer Vision and Geometry Group* (D-INFK).

Bei autonom fliegenden Dronen gibt es aktuell zwei grössere Problembereiche:
Energieversorgung: die aktuellen Batterien reichen für Flugdauer von ca. 25 Min. (bei 60 km/h)
Sicherheit: die Programme, welche die Dronen steuern, müssen eine sehr hohen Qualitätsstufe erreichen, damit sichergestellt ist, dass die Dronen keinen Schaden (z.B. durch Absturz oder Kollision) verursachen. Der Referenz schlägt zu diesem Zweck einen offenen Prozess *open safety* vor, welcher die Erfahrungen aus Open Source übernimmt.

Tom Bujok: *33 things you want to do better*

Ausgehend von der Erkenntnis, dass Software-Code schlecht (oder zumindest verbesserungswürdig) ist, plädiert der Referent dafür, schlechte Gewohnheiten, welche zu schlechtem Code führen, zu identifizieren und so schnell wie möglich zu eliminieren. Schlechte Gewohnheiten können nicht gelöscht, sie müssen überschrieben werden.

Der Referent stellt verschiedene (Java-) Projekte vor, mit deren Einsatz Boilerplate-Code eliminiert werden kann:
Lombok: stellt nützliche Annotationen zur Verfügung, welche den Code vereinfachen (vgl. http://projectlombok.org)
Guava: erleichtert den Umgang mit null, Vorbedingungen (Preconditions), mehrdimensionalen Mengen (Multiset) und Zuordnungen (Multimap, Table) (vgl. http://code.google.com/p/guava-libraries/).
LambdaJ: ermöglicht die Bearbeitung von Collections auf eine pseude-funktionale Weise (vgl. http://code.google.com/p/lambdaj/).
Guice: Leichtgewichtiger Dependency Injection Container (vgl. http://code.google.com/p/google-guice/).
Spock: macht Unit-Tests lesbarer (vgl. http://code.google.com/p/spock/).
Unitils: erleichtert beispielsweise die Erzeugung von temporären Files (zu Testzwecken) und das Lesen des Fileinhalts (vgl. http://www.unitils.org/).
JUnitParams: erleichtert Unit-Tests mit Hilfe von Parametrisierung der Test-Methoden (vgl. http://code.google.com/p/junitparams/).
Awaitility: erleichtert das Testen von asynchronen Systemen (vgl. http://code.google.com/p/awaitility/).
Byteman: (vgl. https://www.jboss.org/byteman).
Groovy: kann beispielsweise in Java eingesetzt werden, um XML mit 2 Code-Zeilen zu parsen.
Grape: Erlaubt, Maven-Abhängigkeiten in Groove einzubinden.
Gradle: Build-Tool mit deutlich kompakterer Konfiguration (verglichen mit Maven).

Martin Naumann: *Mobile web technologies - an overview*

Der Referent berichtete über seine Erfahrungen mit mobilen Anwendungen. Gemäss diesen Erfahrungen ist der DOM auch in Smartphone-Browsern schnell. Es gibt allerdings Operationen, welche für den DOM aufwendig sind. Dies sind vor allem Operationen, welche einen *reflow* auslösen.

Hybride Applikationen: der Referent verwendet für seine mobilen Anwendungen folgenden Technologie-Stack: PhoneGap (Framework); AngularJS (für MVC/MVP); Lungo.js (für UI).

Stefan Saasen: *Real World Git Workflow*

Der Referent behandelte folgende Themen im Zusammenhang mit Git:

Zusammenarbeits-Modell: mit einer dezentralen Versionskontrolle können unterschiedliche Modelle, wie die Repositories miteinander kommunizieren, realisiert werden: *vollständig dezentralisiert* bis *gemeinsam geteiltes Repository*.

Branching-Modell: ein Geschäftsmodell, in welchem kontinuierlich ausgeliefert (*continuous delivery*) wird, verlangt ein anderes Branching-Modell als ein Umfeld, in welchem das Produkt in bestimmten Versionen ausgeliefert wird.

Methoden: Git macht *pull requests* möglich, dies ist ein idealer Zeitpunkt für Code-Reviews. *Forks* von Repositories ist unter Umständen sinnvoll.

Werkzeuge und Automatisierung: An Git-Hooks können Skripte gehängt werden, welche zu bestimmten Punkten (z.B. nach einem Commit) gewisse Aktionen ausführen (z.B. Checkstyle ausführen).

Für mehr Informationen, siehe Präsentation auf http://www.slideshare.net/jazoon13/jazoon-2013-stefan-saasen-real-world-git-workflows.

Empfehlungen:

1. Die Einführung von Git als Versionsverwaltung (als Ersatz von CVS und SVN) sollte geprüft werden.
2. Der Einsatz der von Tom Bujok vorgestellten Java- (bzw. Groovy-) Bibliotheken sollte geprüft werden.
3. Das Einsatzpotential von OpenDolphin (Dierk König) sollte untersucht werden.