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, 13. Oktober 2013 BL

Europython 2013, Konferenz-Bericht

Die Europython 2013 fand wie die beiden vorgängigen Ausgaben in Florenz statt und hatte in etwa die gleiche Grössenordnung: an fünf Konferenztagen wurden in fünf parallelen Tracks mehr als 100 Vorträge, Referate und Präsentationen angeboten.

Die in der folgenden Liste aufgeführten Europython-Präsentationen haben eine gewisse Relevanz für die Software-Entwicklung an der ETH Zürich:

Holger Krekel: *The return of Peer2Peer computing*
Der Referent stellte seinen Vortrag in einen Zusammenhang mit dem NSA-Skandal. Das Internet barg in seinen Anfängen das Versprechen der Freiheit, nun erweist es sich als Mittel des Überwachungsstaats. Die Zukunft des Internets hängt sowohl von der Politik und den Gesetzen ab, aber auch von technologischen Änderungen. Die Überwachung im Internet durch z.B. das NSA ist möglich, weil das Internet entgegen den Annahmen nicht genug dezentral ist, sondern der globale Informationsaustausch durch einige wenige Knoten geführt wird. Um die Freiheit im Internet zurückzugewinnen, müsste das Internet deutlich mehr dezentralisiert werden. Peer2Peer (P2P) – Kommunikation könnte das möglich machen. Auf folgende Punkte müsste Wert gelegt werden: End-to-End-Verschlüsselung, dezentrale P2P routing Topologie, Mediator-ownership. Der P2P-Code müsste Open-Source sein.
Beispielhafte Ansätze: crypto.com, WebRTC (Web Real-Time Communication, d.h. Browser-to-Browser-Streaming).
Des Weiteren plädiert der Referent dafür, dass WLAN von allen Hotspots unverschlüsselt angeboten wird (vgl. Android-App SPAN, Freifunk.net).

Alex Martelli: *„Good enough" is good enough!*
Richard P. Gabriel eröffnete 1989 eine Diskussion zu den Ansätzen „Right Thing“ (aka. “MIT/Stanford approach”) vs. „Worse is Better“ (aka. „Jersey approach“). E. Raymond führte eine analoge Diskussion in „The Cathedral and the Bazaar“ (Kathedrale = “Right Thing” vs. Basar = „Worse is Better”)
Vergleich der Ansätze in Bezug auf:

	Worse is Better	**Right Thing**
Einfachheit	Implementierung UND Interface	Speziell bei Interface
Korrektheit	Besser *Einfach* als *Korrekt*	Höchste Priorität

Konsistenz	Nicht offensichtlich Inkonsistent	Gleicher Stellenwert wie Korrektheit
Vollständigkeit	Kann zugunsten der anderen Punkte geopfert werden	Ähnlich wichtig wie *Einfachheit*

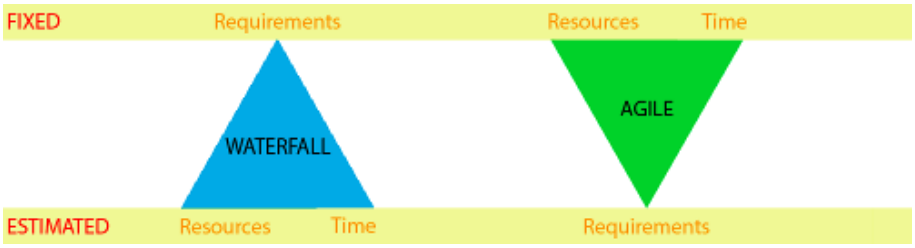
„Right Thing“ ist für Experten, welche alles von Anfang bis zum Ende machen, bevor die User ins Spiel kommen. „Right Thing“ setzt „Bid Design Up Front“ (BDUF) voraus: perfekte Identifikation der Anforderung, führt zu perfekter Architektur, führt zu perfektem Design, führt zu perfekter Umsetzung, braucht sehr viel Zeit! In der realen Welt ändern die Anforderungen etc., eine iterative Vorgehensweise kommt den realen Ansprüchen näher.
Martelli plädiert für den „Worse is Better”-Ansatz, wobei allerdings folgende Ansprüche erfüllt sein müssen: Agiler Prozess mit Versionskontrolle, Code-Reviews, Testing, korrektem Release-Engineering. Der Code-Stil muss klar und elegant sein. Die Dokumentation darf nicht knapp gehalten werden.
Was Up-Front sichergestellt werden muss: Sicherheit (Privatheit, Prüfbarkeit).
„Worse is Better” und Software-Bugs: Fokus auf Fehler, welche zu unwiederbringlichem Verlust oder grossen Reputations-Schaden führen. Aber „Service Recovery Paradox“: Die zufriedensten Kunden sind solche, welche einen Fehler/ein Problem schnell gelöst bekommen.
Beispiele für „Worse is Better” oder „Good Enough“: Unix. TCP/IP vs. ISO/OSI. „Lean Startups“ (vgl. Eric Ries), rasch mit einem minimal brauchbaren Produkt (minimum viable product) an den Start gehen und Reaktionen der Benützer in den nächsten Iterationen verarbeiten.

Ezio Melotti: *Understanding Unicode*
Technisch muss zwischen Characterset (Zeichensatz) und Encoding unterschieden werden. Ein Zeichensatz ordnet jedem Zeichen einen Codepunkt (code point) zu, es ist eine eineindeutige Zuordnung von Zeichen zu Zahlen. Das Encoding ist eine Repräsentation eines Codepunkts in Form von Bytes. Unicode ist ein Zeichensatz, UTF-8 oder UTF-16 sind bestimmte Darstellungen dieses Zeichensatzes. Entsprechend heisst auch die Transformation von Zeichen in Bytes heisst Encoding, die umgekehrte Transformation (Bytes zu Text) heisst Decoding.
Der Unicode-Standard bietet insgesamt 1‘114‘112 Codepunkte an (U+0000 bis U+10FFFF, d.h. 17 Ebenen zu je 2^16, d. h. 65536 Zeichen).
Ein Encoding kann ein oder mehrere Bytes umfassen, ein Multibyte-Encoding kann feste oder variable Länge haben. UTF steht für Unicode Transformation Format. UTF-8 ist variabel (1 od. 2 od. 3 od. 4 bytes), UTF-16 ist variabel (2 od. 4 bytes), UTF-32 ist fix mit 4 bytes.

Thomas Waldmann: *Passwords – the server side*
Anforderungen, die sich aus Passwortschutz ergeben: Passwort muss auf dem Server gespeichert werden und es muss mit der Benutzereingabe (Clear text password) verglichen werden.
Folgerungen:
Verschlüsselung darf nicht symmetrisch sein, d.h. es darf keine Funktion geben, so das CPT = decrypt(P) wo P = encrypt(CPT).
One-way crypte mit md5, apr1, sha1, ntlm sind zu schwach, besser ist cryto-hash mit salt und pepper (z.B. sha256).
GPU-Prozessor kann 10^9 sha256 hashes jede Sekunde berechnen -> brute-force-Attacken
Login-Algorithmus muss so gestaltet sein, dass der (feindliche) Benutzer nicht erkennen kann, ob bei einem gescheiterten Login der Username oder das Passwort falsch war.
Python hat mit passlib eine geeignete Bibliothek für Passwort-Schutz. Passlib ist langsam und konfigurierbar.
Benutzer sollen eine Rückmeldung über die Stärke ihres Passworts bekommen.
Falls der Unfall passiert und die verschlüsselten Passwörter gestohlen werden, müssen rasch geeignete Massnahmen ergriffen werden: Benutzer benachrichtigen, Passwörter zurücksetzen etc.

Russell Sherwood, David Sale: *The "Agile Movement"*

Agile Methoden eignen sich gut in einen kompetitiven Umfeld, weil ein solches Umfeld häufige und rasche Änderungen verlangt.
Jedes Software-Projekt ist durch die drei Randbedingungen Anforderungen, Ressourcen und Zeit bestimmt. Während bei der Wasserfall-Vorgehensweise die Anforderungen von aussen bestimmt werden und die für die Implementierung dieser Anforderung notwendigen Ressourcen und Zeit innerhalb des Projekts geschätzt werden, wird dieses Modell bei den agilen Methoden umgekehrt. Bei agilen Methoden werden Ressourcen und Zeit extern vorgegeben, während die Anforderungen von Projektteam geschätzt werden.



Die agile Vorgehensweise baut auf folgende Prinzipien:

1. Individuals and interactions *over* processes and tools
2. Working software *over* comprehensive documentation
3. Customer collaboration *over* contract negotiation
4. Responding to change *over* following a plan

(Agile Manifesto, 2001)

Die Entwicklungsarbeiten beginnen mit einer “Defuzz”-Phase. In dieser Phase geht es darum, ein konkretes Verständnis für die Anforderungen und deren Umsetzung zu entwickeln, d.h. es werden Eckwerte definiert und getestet. In dieser Phase ist Pair-Programming und ein Test-Driven-Ansatz vorteilhaft. Der Entwickler sitzt mit dem Analysten (und eventuell mit dem Qualitäts-Sicherer) zusammen und entwickelt die Interfaces der Applikation.

Yves J. Hilpisch: *A Better Future with Python*

Python hat einen starken Bezug zu den Wissenschaften, vor allem in den Naturwissenschaften und Finanzwissenschaften wird Python viel eingesetzt. Der wissenschaftliche Alltag wird durch folgende Trinität bestimmt: gesprochene/geschriebene Sprache (Englisch), formale Sprache (Mathematik), Programmiersprache. Python hat eine grosse Nähe sowohl zur formalen Sprache Mathematik wie auch zum Englischen. Ein mathematischer Ausdruck kann sehr einfach und konsistent in Python übersetzt werden. Z.B. „Berechne den inneren Wert einer Option zum Verfallsdatum mit einem Ausführungspreis von 105 für 10‘000 simulierte Preise eines Basiswerts S.“

Mathematik

$$
\begin{aligned}
S_0 &= 100 \\
K &= 105 \\
I &= 10000 \\
S_T^i &\in \{S_T^1, \ldots, S_T^I\} \\
h_T^i &= \max(S_T^i - K, 0)
\end{aligned}
$$

Python/NumPy

```
S0 =100
K = 105
I = 10000
ST = S0 * standard_normal(I)
h = maximum(ST - K, 0)
```

Python bietet für Natur- und Finanzwissenschaftler einen hochinteressanten Stack: Sphinx, IPython Notebook, PythonTEX, dazu NumPy, SciPy, pandas, matplotlib, PyTables. Damit wird es möglich,

ausführbaren Code in das Dokument einzubetten und die Resultate automatisch im Dokument erscheinen zu lassen. Dies erlaubt *offene Forschung* mit reproduzierbaren Resultaten für alle interessierten Leser.

Das Potential von Python ist auch der amerikanische Regulierungsbehörde *Securities and Exchange Commission* (SEC) aufgefallen. In einem Proposal hat die SEC 2010 entschieden, den Einsatz von Python vorzuschreiben, um den Vermögensfluss auf Grund der Kontrakte exakt zu beschreiben, damit auf diese Weise die Fähigkeit der Investoren erhöht werden kann, die Qualität und Mechanismen von *Asset Backed Securities* (ABS) zu erfassen.

Anders Lehmann: *Using IPython Notebook in the Classroom*

Im Unterrichtseinsatz wird die Kombination von Text, Bild und Interaktivität (ausführbarer Code) geschätzt. Diese Kombination wird mit *IPython Notebook* auf einfache Art geliefert.
IPython ist eine Python-Version, welche im Browser läuft.
IPython Notebook ist eine Erweiterung zu IPython, mit welcher Python-Code in eine Webseite eingebettet werden kann. iPython Notebook ist wie MatLab im Browser. Für IPython Notebook gibt es Physik-Erweiterung (fügt physikalische Einheiten und Physik-Konstanten zu Python, überprüft Übereinstimmung von Einheiten).
IPython benützt Markdown für Textauszeichnung.
Problematisch an der aktuellen Version von iPython Notebook ist, dass die Änderungen nicht automatisch gespeichert werden. Wird die Browser-Seite unaufmerksam oder unbeabsichtigt geschlossen, können Änderungen verlorengehen.

Jacob Hallén: *Testing for Beginners*

Verschiedene Tests:

Phase	**Rolle**	**Test-Typ**
Kundenwünsche		
	Analyst	Akzeptanz-Tests Deplyoment-Tests
Anforderungen		System-Tests
	Programmierer	Unit-Tests Integrations-Tests
Programm/Applikation		

Warum testen?

- Formulieren, was das Code-Teil tun soll -> Schnittstelle testen
- Überprüfen, dass wir machen, was der Test verlangt
- Fehlerquellen eliminieren
- Verständnis des Code für die Zukunft (im Hinblick auf Refactoring) verbessern
- Bei testgetriebener Entwicklung (TDD) signalisiert ein fehlerfreies Testresultat, dass der Code aus Sicht der Anforderungsspezifikationen korrekt und vollständig ist

Was soll getestet werden?

- Speichern und Auslesen von Informationen (in Datenmodell)
- Funktionalität: entspricht das Resultat einer Berechnung dem erwarteten Resultat
- Systemgrenzen, z.B. zur Persistenz-Schicht. Das ist häufig anspruchsvoll und kann mit Mock-Objekten erleichtert werden.

Ein Test ist ein Beispiel für einen bestimmten Fall. Es können nicht alle Fälle getestet werden. Eine gute Vorgehensweise ist: ein charakteristischer Fall testen, alle Grenzfälle testen, alle expliziten Exceptions testen.
Je kleiner die Test-Abdeckung ist, desto grösser ist die technische Schuld (d.h. der zusätzliche Zeitaufwand und die Extra-Kosten, die in der Zukunft notwendig werden, weil keine Tests beim Entwickeln des Codes geschrieben worden sind).
Wenn keine vollständige Test-Abdeckung geleistet werden kann, so müssen die grundlegenden Fälle in jedem Fall getestet werden.

Anregungen:

- Kennen die Dozenten und Wissenschaftler an der ETH das Potential von *IPython Notebook*? Eventuell sollte mit dem LET abgeklärt werden, ob mit geeigneten Veranstaltungen (z.B. Präsentationen, Kursen) die möglichen Benützer auf dieses Werkzeug aufmerksam gemacht werden sollen.

- Soll die Einführung einer *Scrum-Defuzz*-Phase für die Software-Entwicklung bei den ID-SWS in Erwägung gezogen werden?
  Mir scheint eine solche Vorgehensweise sinnvoll zu sein. Wenn möglich sollten an einem geeigneten neuen Projekt mit TDD und Pair-Programming konkrete Erfahrungen gesammelt werden.

Benno Luthiger, Oktober 2013