Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Informatikdienste
Software Services

ETH Zürich
Benno Luthiger
STC E 13
Stampfenbachstrasse 67
8092 Zürich

Telefon +41 44 632 57 65
benno.luthiger@id.ethz.ch
www.foss.ethz.ch

Software Services (SWS)
Informatikdienste
ETH Zürich

Zürich, 24. August 2015

EuroPython 2015, Konferenz-Bericht

Die EuroPython 2015 fand in Bilbao statt. An den fünf Tagen der Hauptkonferenz wurden in mehreren 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:

Dmitry Trofimov: *Can Rust make Python shine?*

Folien: https://dl.dropboxusercontent.com/u/10672619/Europython%202015%20-%20Can%20Rust%20Make%20Your%20Python%20Shine_.pdf

Rust ist eine Programmiersprache der Mozilla-Foundation. Rust soll ähnlich schnell sein wie C++, aber jegliche (Sicherheits-) Probleme, welche auf Grund Speicherzugriffsfehlern und Pufferüberläufen (segfaults) geschehen, eliminieren.

Mit Rust FFI (*foreign function interface*) kann Rust in Python eingebunden werden. Damit ist es beispielsweise möglich, einen Python-Profiler zu schreiben.

Núria Pujol, Ignasi Fosch: *What dojos are and how we run them at pyBCN*

Folien: https://docs.google.com/presentation/d/1Jvl4fWJxhcFm8qXK_fGs-gFN79z1PTr8wb6kouK6zjE/edit#slide=id.gb4c5bee4b_0_132

Coding Dojos bieten eine alternative Möglichkeit, das Wissen über und die Erfahrung mit einer Programmiersprache zu erweitern. Coding Dojos sind speziell gut geeignet, Fachwissen von erfahrenen Personen an Einsteiger weiterzugeben.

In einem Coding Dojos sind immer zwei der Teilnehmer am Programmieren (*pair programming*), wobei der eine die Tastatur bedient (*Driver*) und der andere (*Navigator*) Ideen und Ratschläge beisteuert. Entwickelt wird

gemäss *Test Driven Development* (TDD), d.h. es wird kein Code ohne vorhergehenden Test erzeugt. Nach 3-8 Minuten wechselt der Driver ins Publikum zurück, der vorherige Navigator übernimmt die Tastatur, an seiner Stelle kommt ein neuer Navigator aus dem Publikum. Nachdem jeder Teilnehmer einmal Driver und Navigator war, spätestens aber nach 25 Min. wird einer Retrospektive abgehalten. Danach beginnt ein neuer Zyklus, auch *Randori* genannt.

Das Programmier-Problem, welches in einem Coding Dojo gelöst werden soll, wird *Kata* genannt.

Radoslaw Jankiewicz: *Writing quality code*

Folien: http://www.slideshare.net/radek_j/euro-python-2015-writing-quality-code

Die Qualität von Code kann grob nach externen oder nach internen Kriterien beurteilt werden.

Externe Kriterien: Korrektheit, Benutzerfreundlichekeit (Usability), Effizienz, Zuverlässigkeit, Integrität, Anpassbarkeit, Präzision, Stabilität.

Interne Kriterien: Wartbarkeit, Flexibilität, Übertragbarkeit, Wiederverwendbarkeit, Lesbarkeit, Testbarkeit, Verständlichkeit.

Die Verständlichkeit von Code ist sehr wichtig. Untersuchungen zeigen, dass Programmierer deutlich mehr Zeit damit verbringen, den Code anderer Entwickler zu verstehen, als Code zu schreiben oder zu modifizieren.

Metriken zur Bestimmung von Software Qualität:

Cyclomatic Complexity: jedes *if* oder *for* steigert die Komplexität um einen Punkt. Für jeden Punkt sollte ein eigener Unit-Test bereitgestellt werden.

Halstead: Anzahl Operatoren und Operanden.

Wartbarkeits-Index (*Maintainability Index*): Kombination von Cyclomatic Complexity, Halstead und Anzahl Codezeilen.

Mit dem Python Radon-Modul kann die zyklomatische Komplexität und der Wartbarkeits-Index von Python-Code gemessen werden.

Andreas Dewes: *Code is not text! How graph technologies can help us to understand our code better.*

Folien: http://de.slideshare.net/japh44/code-is-not-text-how-graph-technologies-can-help-us-to-understand-our-code-better

Wird Programm-Code als Graph interpretiert, können mit den für Graphen entwickelten Werkzeugen einfach Code-Probleme und –Zusammenhänge visualisiert werden.

Vergleich von Code als Text vs. Graph:

Text: einfach zu schreiben (+), einfache Anzeige (+), universelles Format (+), kompatibel (+), nicht normalisiert (-), schwierig zu analysieren (-)
Graph: einfach zu analysieren (+), normalisiert (+), einfach zu transformieren (+), schwierig zu erzeugen (-), noch nicht kompatibel (-)

Marco Buttu: *Lessons learned about testing and TDD*

Folien: http://marco-buttu.github.io/pycon_testing/

Gewonnene Erfahrungen:

- Wenn Tests nicht nutzlos oder schädlich sein sollen, müssen sie fehlschlagen können.
- Bevor ein Fehler geflickt werden soll, muss ein Test erzeugt werden, welcher den problematischen Fall testet und fehlschlägt.
- Integrations-Tests können eingesetzt werden, um den Vertrag zwischen Komponenten-APIs zu begründen.
- Unit-Tests müssen schnell und selektiv sein. Im Idealfall enthalten sie eine Assert-Anweisung.
- Falls die Schnittstelle zu externen Ressourcen stabil ist, ist es besser, Simulatoren anstatt Mock-Objekte zu verwenden.
- TDD gewährleistet eine maximale Testabdeckung.
- Es gibt kein Ansatz, welcher für jeden Kontext gleich gut passt.

Matt Bennett: *Nameko for Microservices*

Folien: http://mattbennett.github.io/presentation-microservices-nameko-europython15/

Konzept von Mikro-Service: Eine Anwendung wird in einzelne Dienste aufgeteilt, welche als separate Prozesse ausgebreitet werden können. Damit soll vor allem die Skalierbarkeit der Anwendung verbessert werden.

Im Gegensatz zu einer monolithischen Anwendung gilt für einen Mikro-Service das ACID-Prinzip nicht mehr. Stattdessen wird BASE (Basically available, Soft-state, Eventually consistent) umgesetzt. Der Nachteil von Mikro-Service ist, dass dies zu grösserer Komplexität im Betrieb führt.

Das Python-Framework Nameko ist speziell dafür ausgelegt, Mikro-Service zu schreiben.

Kyran Dale: *Data-visualisation with Python and Javascript: crafting a data-viz toolchain for the web*

Folien: http://kyrandale.com/static/talks/reveal.js/index_pydata2015.html

Für die Visualisierung von Daten im Browser gibt es keine Alternative zu JavaScript.

D3 (http://d3js.org/) ist eine JavaScript-Bibliothek, welche mit wenig Aufwand schöne Visualisierungen möglich macht.

Der Redner präsentiert seine Kette von Werkzeugen, mit welchen Daten aufbereiten und danach visualisiert werden können. Beispiel: die Liste aller Nobelpreisträger ausgehend von der Wikipedia-Seite visualisieren:

- Daten auslesen mit Scrapy
- Daten aufbereiten mit Pandas (Python Data Analysis Library)
- Daten über eine Rest-Schnittstelle anbieten mit Flask
- Daten im Browser darstellen mit D3.

Marc-Andre Lemburg: *Python idioms to help you write good code*

Die in PEP-8 festgelegten Code-Konventionen sollten eingehalten werden. Sie können mit dem pep8-Package überprüft werden.

Weitere Werkzeuge, mit welchen der Code überprüft werden kann: pylint, pyflakes, flake8.

Radoslaw Jan Ganczarek: *Code Quality in Python - tools and reasons*

Folien: http://www.slideshare.net/RadosawJanGanczarek/code-quality-in-python-tools-and-reasons

Holger Krekel: *Towards a more effective, decentralized web*

Die Möglichkeiten des Internets, d.h. die Knotenstruktur, welche direkten Austausch zwischen Peers (peer2peer) zulässt, sollten besser genutzt werden. Bei einem File-Transfer von einem Gerät zu einem anderen sollte beispielsweise nicht über einen fernen Server gekehrt werden.

Je mehr mobile Geräte verwendet werden, desto häufiger wird die Erfahrung, dass diese Geräte temporär offline sind. Die Kommunikation mit und die Verwendung von solchen Geräten soll von einem modernen Netzwerkprotokoll bedacht werden (z.B. bittorrent, git, vgl. auch http://offlinefirst.org/). Alle Implementierungen solcher Ansätze basieren auf Merkle-Bäumen (Hash-Trees).

Das *Interplanetary File System* (IPFS) ist ein Konzept für ein modernes, dezentralisiertes Web (vgl. http://ipfs.io/). IPFS kombiniert die Konzepte von BitTorrent, Git, Merkle-Bäumen und der Bitcoin-Blockchain. Der Zugriff auf Inhalt erfolgt über Inhalts-adressierte Hyperlinks. Beispiel: ipfs://<*content-hash*>/ep2015.

Sebastian Neubauer: *A Pythonic Approach to Continuous Delivery*

Folien: https://ep2015.europython.eu/media/conference/slides/a-pythonic-approach-to-continuous-delivery.pdf

Bei continuous delivery geht es darum, die Mauern zwischen den Silos von Entwicklung und Betrieb niederzureissen -> DevOps.
Das Ziel ist, den Prozess der Release-Bildung und Inbetriebnahme zu automatisieren. Kann das erfolgreich Umgesetzt werden, werden Entwickler-Kapazitäten frei, um von Kunden gewünschte Features schneller zu Realisieren. Gleichzeitig werden Sysadmin-Kapazitäten frei, um den Betrieb (7 * 24, Performanz) zu optimieren.

Vgl. Buch von Jez Humble "Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation"

Guido van Rossum: *Type Hints for Python 3.5*

Vorteil von Typ-Hinweisen in Python: Fehler können früher gefunden werden (von IDE), Dokumentationshilfe für IDE. Typ-Hinweise sind ab Python 3.5 und optional, d.h. Python bleibt eine dynamische Skriptsprache.
‚Type' vs. ‚class': Typen sind für type checker, class ist für die Runtime.

Stephan Erb: *Release Management with Devpi*

Folien: https://ep2015.europython.eu/media/conference/slides/release-management-with-devpi.pdf

Devpi ist ein auf Python basierender Repository-Server, der mit Nexus verglichen werden kann.
Für ein stabiles Release-Management ist wichtig, dass die Release-Pakete unveränderlich sind.
Lösungsmöglichkeit: Versionsnummer von Git-Versions-Tag ableiten.

Nicholas Tollervey: *Lessons learned with asyncio ("Look ma, I wrote a distributed hash table!")*

Folien: http://ntoll.org/static/presentations/asyncio/index.html

Eine verteilte Hash-Tabelle (distributed hash table, DHT) ist ein Peer-to-peer-Datenspeicher. Die Python-Bibliothek asynchio erlaubt es, solche modernen Konstrukte umzusetzen.

Patrick Mühlbauer: *Building nice command line interfaces - a look beyond the stdlib*

Python-Skripte eignen sich gut als Skripte, welche von der Kommandozeile aufgerufen werden. Auch solche Skripte sollten benutzerfreundlich sein. Am wichtigsten in Hinsicht auf die Benutzerfreundlichkeit ist die Dokumentation der Argumente, mit welchen die Skripte aufgerufen werden. Folgende Bibliotheken sind mächtige und komfortable Alternativen zu den Möglichkeiten, welche in Python stdlib zur Verfügung stehen:

- Click: http://click.pocoo.org (arbeitet mit Dekoratoren)
- docopt: http://docopt.org (arbeitet mit Doc-Strings)
- Cliff: http://docs.openstack.org/developer/cliff

Carrie Anne Philbin: *Designed for Education: A Python Solution*

Folien: https://speakerdeck.com/missphilbin/designed-for-education-a-python-solution

Warum sollte Programmieren in der schulischen Ausbildung (ab 6 Jahren) einen hohen Stellenwert einnehmen? Kinder sind kreativ und frustrationstolerant; gute Kenntnisse im IT-Bereich, spezielle in Programmieren, fördert die soziale Mobilität; eine breite Ausbildung in IT fördert die Vielfalt in technischen Bereich; bei zunehmender Automatisierung braucht es vermehrt Programmierer.

Python-Bibliotheken, welche für den Einstieg mit Kindern geeignet sind:
Pygame Zero, a zero-boilerplate game framework for education (http://mauveweb.co.uk/posts/2015/05/pygame-zero.html)

Geeignete Entwicklungsumgebungen sind ein Problem für den Schulunterricht. Viele gute IDEs sind zu kompliziert, wenn es darum geht, ein neues Skript (File) zu erzeugen und im Filesystem zu speichern. Sonic Pi (http://sonic-pi.net/) ist in dieser Hinsicht ein interessanter Ansatz.

Ben Nuttall: *Physical computing with Python and Raspberry Pi*

Folien: https://ep2015.europython.eu/media/conference/slides/physical-computing-with-python-and-raspberry-pi.pdf

Ein Raspberry Pi eignet sich vorzüglich, um mit Python interaktive, physische Systeme durch die Verwendung von Hardware und Software zu erstellen (sog. *physical computing*).

Rafal Nowicki: *BDD: You're doing it wrong!*

Behaviour driven development (BDD) ist eine Technik der agilen Softwareentwicklung. Bei BDD werden während der Anforderungsanalyse die Aufgaben, Ziele und Ergebnisse der Software auf eine Art textuell festgehalten, dass die mit diesen Anforderungen erzeugte Software später automatisiert getestet (d.h. auf ihre korrekte Implementierung geprüft) werden kann. Ein häufig benutztes Format für die BDD-Beschreibungssprache ist Gherkin (https://github.com/cucumber/cucumber/wiki/Gherkin). Bei automatisiertem Testen wird Gherkin zusammen mit Selenium verwendet. Im Java-Umfeld wird FIT bzw. FitNesse mit BDD verwendet.

Alex Willmer: *Taking the pain out of passwords and authentication*

Folien: https://moreati.github.io/passwordspain/

Passwörter stellen ein Problem dar. Die Benützer lieben einfache Passwörter, die Server fordern komplizierte Passwörter. Aus dieser Kluft resultiert häufig, dass der Benützer als schwächstes Glied versagt und zum Einlass für Hacks und Datendiebstahl wird.
Die aktuellen Alternativen zu Passwörter (federated logins, hardware tokens, Phones/Apps, Biometrie) haben neben Vorteilen auch gewichtige Nachteile.
Statt Passwörter sollten Passphrasen verwendet werden. Diese sind einfacher zu merken und können auf dem Mobilgerät einfacher eingetippt werden.
Muss ein Passwort gewählt werden, soll eine visuelle Rückmeldung über die Stärke des Passworts gegeben werden.

Perspektive: fido-Allianz (https://fidoalliance.org/specifications/overview/), will UAF (Universal Authentication Framework) und U2F (Universal Second Factor) als Standard einsetzen. Diverse wichtige Firmen (Microsoft, Google, PayPal) machen bei diesem Konsortium mit.

Mandy Waite: *So, I have all these Docker containers, now what?*

Folien: https://ep2015.europython.eu/media/conference/slides/keynote-so-i-have-all-these-docker-containers-now-what.pdf

Dank Containern wie Docker kann eine gute Isolierung der Applikation von anderen Applikationen erreicht werden. Das Betriebssystem wird geteilt, dagegen werden die Bibliotheken pro Applikation gepackt werden. Trotz Isolierung ermöglicht dies eine gute Performanz.

Zur Verwaltung der Docker-Container hat Google Kubernetes entwickelt (http://kubernetes.io/).

Scott Triglia: *Arrested Development - surviving the awkward adolescence of a microservices-based application*

Folien: https://ep2015.europython.eu/media/conference/slides/arrested-development-surviving-the-awkward-adolescence-of-a-microservices-based-application.pdf

Mikro-Service sind ein Ansatz, um ein Applikation in Form einer Gruppe von schlanken Dienste umzusetzen, welche alle in ihren eigenen Prozessen laufen und die über leichtgewichtige Prozesse kommunizieren.

Mikro-Service erlauben Entkopplung, was allerdings zu zusätzlicher Komplexität bei der Kommunikation der einzelnen Service führt.
Schnittstellen sind die Summe der APIs, der geteilten Bibliotheken und der Daten, welche durch die Schnittstellen fliesst.
Mit Mikro-Service muss teilweise DRYness geopfert werden.
Service-Schnittstellen sind eine gute Gelegenheit, bewusst Systeme zu entkoppeln.
Schnittstellen müssen bewusst und explizit sein. Swagger (http://swagger.io/) ist ein gutes Produkt, um Schnittstellen zu beschreiben.
Logging ist elementar für die Fehlerbehebung. Mit gutem Logging und Monitoring können Fehler erkannt und behoben werden, bevor solche Probleme vom Benützer erkannt werden. Zu diesem Zweck müssen die Log-Meldungen intelligent visualisiert werden. Logstash (https://www.elastic.co/products/logstash) ist ein nützliches Hilfsmittel dazu.

Schlussbemerkungen / Anregungen:

- Die *DevOps*-Perspektive sollte für die ID ernsthaft geprüft werden. Dabei müssen folgende Fragen geklärt werden:
  - Können mit *Continuous Delivery* unsere Kunden mit ihren Wünschen besser (schneller, mit besserer Qualität) bedient werden?
  - Haben wir bei den ID die fachliche Kompetenz, den Lebenszyklus einer Applikation von der Entwicklung bis in den Betrieb maximal zu automatisieren?
  - Sind wir bei den ID fähig, moderne Deployment-Konzepte wie Docker einzusetzen?
  - Sind wir fähig, über die Abteilungsgrenzen zwischen Entwicklung und Betrieb fachlich zusammenzuarbeiten? Die Entwickler können beispielsweise den Sysadmins zeigen, wie man gute Skripte (für Automatisierung von Task) erstellt und versioniert. Die Sysadmins können den Entwickler zeigen, wie man die Programme optimal packt, damit sie vollautomatisiert installiert werden können und automatisch im Applikations-Monitoring erscheinen.
- Innerhalb der Software-Entwicklung sollten neue Formen der Wissens- und Erfahrungs-Vermittlung wie Code Katas und Coding Dojos ausprobiert werden.

Luthiger Benno (ID SWS)