

Fail Fast to Succeed

Der Wert von Feedbackschleifen

Christoph Bonitz

Mai 2019

Warum dieser Vortrag?

- Bogen spannen über die Themen der Lehrveranstaltung aus der Perspektive eines Praktikers
- Idealerweise: Kontext + Motivation
- Notwendigerweise subjektiv
 - Weniger rigoros und wissenschaftlich als reguläre Vorlesung
 - Meinungen sind meine, nicht die meines Arbeitgebers
- Bitte jederzeit (hinter)fragen!

Agenda

WARUM FEEDBACK WICHTIG IST

FEEDBACK AUF INDIVIDUELLER EBENE

PROJEKTE AUF SCHNELLES FEEDBACK TRIMMEN

SCHNELLE RÜCKMELDUNGEN WÄHREND DES PROGRAMMIERENS

SCHNELLE RÜCKMELDUNGEN FÜR DAS TEAM

FEEDBACK FÜR DAS PRODUKT

Warum Feedback wichtig ist

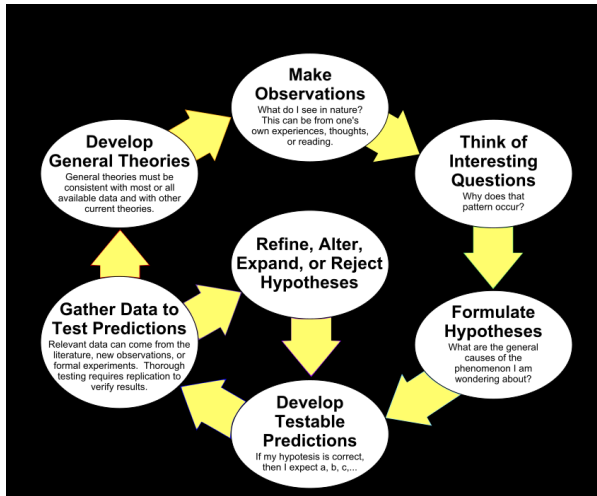
- Wir verstehen unsere Umwelt durch Interaktion und Rückmeldungen
- Berufsleben: Ziel Erfolg, nicht Bewertung am Projektende
- Keine (Lebens-) Zeit für sinnlose Projekte
- Früher korrigieren = billiger
- Fehlervermeidung noch billiger

Kernthese: **Wir müssen uns aktiv um Feedback auf jeder Ebene bemühen.**

- 😊 Viele einfache & billige Methoden

Erfolgsgeschichte Wissenschaft

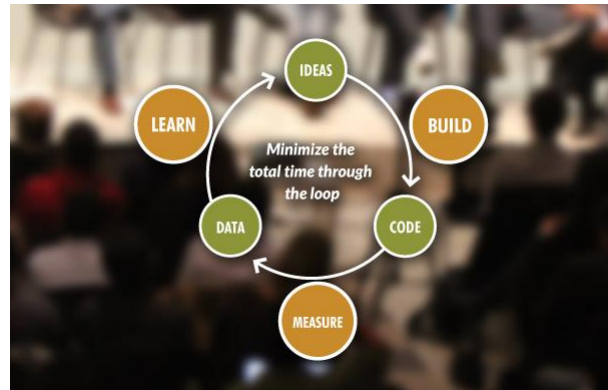
- Etablierung der *Wissenschaftlichen Methode* hat das Fortschreiten der Wissenschaft massiv beschleunigt.
Schlüssel: Empirisches Arbeiten



Quelle: Wikimedia Commons

Erfolgsgeschichte (Lean) Startups

- Product/Market Fit
(Steve Blank: The Four Steps to the Epiphany, 2006)
- Landing Pages ohne Produkt dahinter
- Elevator Pitch
- Minimum Viable Product
- Pivoting
- A/B Testing
- Analytics



<http://theleanstartup.com/principles>

Erfolgsgeschichte Fliegerei

- Crew Resource Management
- Unfall => Suche nach Verbesserungen (nicht Schuld)
- Lernen auf Flugzeugen mit niedrigen Betriebskosten
- Simulatoren (Lehre, Checks)



Quelle: The Economist

Persönliches Feedback

ANNEHMEN

GEBEN

Persönliches Feedback annehmen

- Wertschätzung: Erst zuhören, dann reagieren
 - Technik: Bedanken & Paraphrasieren
- Nicht immer angenehm
 - Will ich lieber nichts hören aber dann keinen Erfolg haben?
 - Will ich, dass mit mir oder hinter meinem Rücken geredet wird?
 - Auch nicht konstruktives Feedback kann mir helfen.
- Feedback einfordern
 - Leichter von vertrauten Menschen
 - Informelle Gespräche vor großen Meetings
- Auf Feedback reagieren = Schlüsselkompetenz
 - Direkte Kommunikation als Differenziator zu Offshoring

Persönliches Feedback geben I

- Höflich und wertschätzend
- Kulturelle Normen beachten
- Regelmäßig positives Feedback geben
- Negatives Feedback unter vier Augen
 - Ausnahme: Übergriffiges Verhalten sofort ansprechen und sichtbar machen.
- Die eigene Meinung aussprechen (nicht „Wahrheit“)
- Über das Problem, nicht die Person sprechen
 - *Der Code war für mich schwer zu lesen (nicht: Dein Code ist...)*

Persönliches Feedback geben II

- Alternativen anbieten
 - Wie wäre es, wenn wir hier einen Kommentar einfügen?
 - Könnten wir die Variablen leichter verständlich benennen
- Nicht erst beginnen, wenn es ein Problem gibt
 - Regelmäßig positive Rückmeldung geben
 - Feedback institutionalisieren, z.B. regelmäßige Code-Reviews, Retrospektiven, 1:1 Meetings

Weiterführende Hinweise

- Soft Skills-Lehrveranstaltungen nutzen
- *Miteinander Reden* Reihe
(Friedmann Schulz v. Thun et al.)

Projekte auf schnelles Feedback trimmen

ITERATIVE ANSÄTZE

FEEDBACK IN DEN PROZESS EINBAUEN

DATEN SAMMELN UND VERFOLGEN

Iterative Ansätze – Bsp. Scrum

- Kurze Iterationen (1-4 Wochen)
 - Verhindert, unangenehme Erkenntnisse vor sich herzuschieben
- Potentially shippable product increment
 - Nichtfunktionale Anforderungen immer erfüllt
- Review Meetings mit Projekt-Stakeholder (Kunden!)
- Retrospektiven
 - Lernen aus dem letzten Sprint
- Bevorzugung von co-located Teams
 - Persönliche Kommunikation

Feedback in Prozess einbauen I

- Prototypen
 - Am besten Throwaway Prototyping
 - Alles außer den Kern des Problems weglassen
- Code Reviews, Pair Programming
 - Man übt regelmäßiges Feedback (geben/nehmen)
 - Man lernt schnell von anderen. Subjektiv: massive Verbesserung
 - Wichtig: JedeR reviewt jedeN
- Architektur nicht top-down aus dem Elfenbeinturm
 - RfCs
 - Arbeitsgruppen
 - Beispielcode (*Exemplar*)

Feedback in Prozess einbauen II

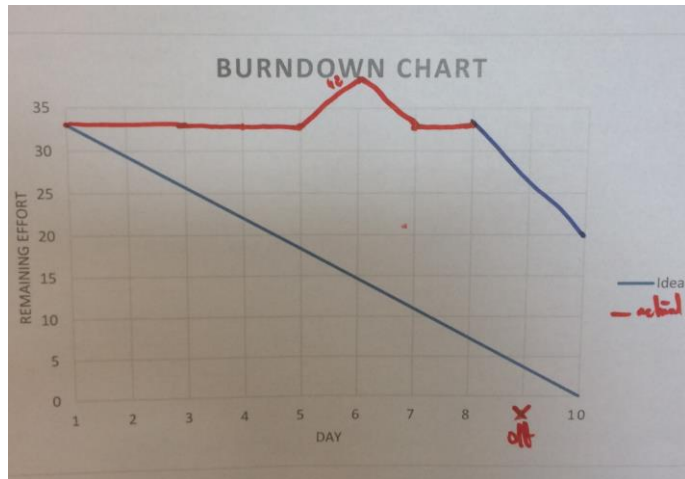
- Neue Features sofort testen
 - Am besten auch durch Kunden
- Usability Tests
 - Paper prototypes
 - Hallway usability testing
 - Formal usability testing

Daten sammeln und verfolgen I

- Kein Projekt ohne Issue-Tracker!
- Ein Ticket für
 - Jedes Problem
 - Jede Story
 - Jede Infrastrukturänderung?
 - Sonstige TODOs?
- Nachverfolgbarkeit
- Vorteil: Auswertungen über längere Zeit
 - Auch im Nachhinein möglich
 - Diskussionen auf empirischer Basis
 - Interventionen überprüfen

Daten sammeln und verfolgen II

- Auch an analoge Methoden denken
 - Burndown
 - Stimmungsbarometer
 - Stimmungskurven



Während des Programmierens

EFFEKTE SOFORT SEHEN (NICHT IMMER MÖGLICH)

EFFEKTE SCHNELLER SEHEN

Beispiel: VR Live Coding (John Carmack)

- <https://www.youtube.com/watch?v=ydyztGZnbNs>
- Programmierumgebung optimiert auf gute Performance und schnelle Iteration
 - Skriptsprache (LISP-Dialekt)
 - Asynchron zu main rendering loop
 - Effekte sofort sichtbar

Beispiel: Distill

- Niveau eines wissenschaftlichen Papers
- Interaktive Teile zum besseren Verständnis
- <http://distill.pub/2017/momentum/>
- Inspiriert von Ideen von Bret Victor
(<http://worrydream.org>)
 - Siehe auch: <http://lighttable.com/> - Versuch, diese Prinzipien in einer IDE umzusetzen

Beispiel: Smalltalk

- Bereits in den 80er Jahren
- Programmier/Ausführungsumgebung vermischt
- System ist zur Laufzeit manipulierbar
- <https://www.youtube.com/watch?v=JLPiMI8XUKU>
 - Randnotiz: hier ist sogar die Maus noch neuartig genug, um extra hergezeigt zu werden
- Zum Ausprobieren: Squeak <http://squeak.org/>

Und was machen die anderen?

- Nicht jede Art von Software ermöglicht sofortiges Feedback
 - Schon gar nicht von Haus aus
- Wenn möglich, Technologien mit kurzem Turnaround bevorzugen
- Aktiv Feedbackschleifen schaffen

IDE, beste Freundin der EntwicklerInnen

- Erinnerungen: Java im Texteditor
- Moderne IDEs bieten in Echtzeit u.a.
 - Code completion
 - Statische Code-Analyse
 - Verwendungssuche
 - Rendering von Dokumentation
 - Kompilation beim Speichern
 - Hot code replacement (mehr oder weniger gut)
 - Refactoring-Unterstützung
 - Codegenerierung
- Es zählt sich aus, eine IDE *gut zu beherrschen*

Unit Testing

- Klassisch – *Red/Green Cycle* des *Test Driven Development*
- Heute in manchen Sprachen: test on save
 - Permanent laufender Prozess hört auf Änderungen im Dateisystem und startet die Tests
 - Ergebnisse werden sofort angezeigt
- Unit Tests helfen, den Code zu strukturieren
- Unit Test helfen, die Intention hinter Code auszudrücken
- Unit Tests helfen (zusammen mit Integrationstests), sicherer zu refaktorisieren

Testability

- Heute heben Frameworks oft heraus, wie leicht und gut Code Unit-testbar ist. Bsp: angular.io:

Full Development Story

Testing

With Karma for unit tests, you can know if you've broken things every time you save. And Protractor makes your scenario tests run faster and in a stable manner.

Startup-Zeit der Anwendung verkürzen

Wie lange muss ich zwischen einer Änderung und Ausprobieren der Effekte warten?

- Rentiert sich schnell:
 - Zig mal pro Tag
 - Flow nicht unterbrechen mindestens so wichtig wie die verlorene Zeit
- Situation unterschiedlich je nach Sprache
- Bei Anwendungen mit Login-Screen: Kann ich diesen für die Entwicklung deaktivieren (ohne Sicherheitslücken einzubauen!!!)?

Interaktive Umgebungen nutzen (speziell für Prototypen)

- Beispiel: Jupyter Notebooks:
Literate Programming für Julia,
Python, R
- Nichts davon ist komplett neu
 - Literate Programming vorgestellt
von D.E. Knuth in 1980ern
 - Lisp REPLs gibt es schon sehr
lange (kein genaues Datum
gefunden)
 - Smalltalk Umgebungen sind sehr
interaktiv (1980er)

Import the data

This reads and merges the two dataframes.

```
In [2]: a = read_perdata('c:/dev/perdata/a/')  
b = read_perdata('c:/dev/perdata/b/')
```

A first exploration

```
In [3]: print(a.describe())  
print(b.describe())
```

	time	db	ratio
count	999986.000000	999986.000000	999986.000000
mean	386.522565	191.757300	0.264470
std	2387.021478	2040.350488	0.220254
min	1.000000	0.000000	0.000000
25%	16.000000	2.000000	0.009415
50%	28.000000	6.000000	0.166667
75%	152.000000	39.000000	0.389053
max	430848.000000	425480.000000	0.999641

	time	db	ratio
count	999985.000000	999985.000000	999985.000000
mean	230.050503	95.040884	0.207001
std	1182.147181	552.142187	0.146171
min	8.000000	1.000000	0.001582
25%	18.000000	3.000000	0.176471
50%	21.000000	5.000000	0.230095
75%	28.000000	9.000000	0.357143
max	128067.000000	91786.000000	0.999857

Histograms of time, db and ratio

```
In [4]: def gtime_t10s(df):  
        return df[(df['time'] < 10000) & (df['time'] > 0)]  
subset_a = gtime_t10s(a)  
subset_b = gtime_t10s(b)
```

Resultate visualisieren

- Code instrumentisieren, Visualisierung der Resultate
- Umgebung visualisieren
 - JVisualVM
 - Resource Monitor / Task Manager / htop
- *Application Performance Monitoring/Management*

Feedback für das Team

CONTINUOUS BUILD

STATISCHE CODEANALYSE

CONTINUOUS INTEGRATION

CONTINUOUS DELIVERY

CONTINUOUS INTEGRATION BEI CA

Continuous Build I

- Voraussetzung: Command Line Build
 - Standardtools!
 - Makefiles / Autotools
 - Ant / Maven / Gradle
 - Grunt / Gulp / ...
 - Managed dependencies: Möglichst wenig Abhängigkeit von Umgebung
 - Sollte möglichst das Endprodukt bauen
 - Build nur erfolgreich, wenn Unit Tests laufen

Continuous Build II

- Build on Commit/Push
 - Single Source of Truth statt *works on my machine*
 - Tools wie Jenkins, Travis CI, Teamcity, ...
 - Standardisierte Umgebungen

Jenkins Pipelines Administration Logout

Pipelines New Pipeline

Favorites

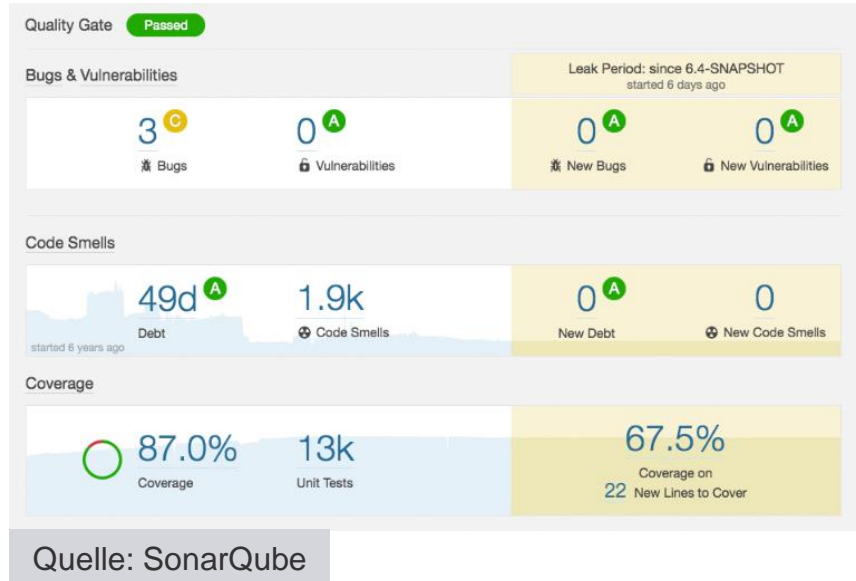
- bitwise-jenkins / git-plugin master a few seconds ago
- bitwise-jenkins / junit-plugin blog/blue-ocean-editor #d3d9a39 12 days ago
- bitwise-jenkins / junit-plugin PR-7 #b518858 12 days ago

Name	Health	Branches	PR
bitwise-jenkins / git-plugin		-	
bitwise-jenkins / hermann		-	
bitwise-jenkins / JS-Nightwatch.js		-	
bitwise-jenkins / junit-plugin		1 passing	2 passing
bitwise-jenkins / logstash-plugin		-	

Quelle: Jenkins.org

Statische Codeanalyse

- Beispiel: SonarQube <https://www.sonarqube.org/>
 - Regeln
 - Maven Plugin
 - History
 - Quality Hotspots
 - IDE Plugins



Continuous Integration

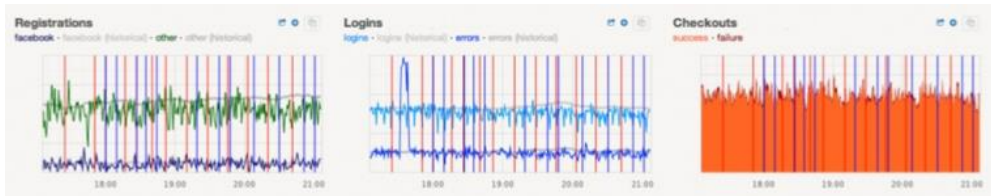
- Nach dem Build:
Gesamte Anwendung zentraler Stelle deployen
- Single Source of Truth
- Automatisierte Integrationstests
- Sehr praktisch für manuelle Tests: Ich kann Changes testen, die ich noch nicht auf meinem Rechner habe (z.B. anderer Branch)
- In der Praxis nicht immer trivial – mehr dazu später

Automatisierte Integrationstests

- Testen das System als Ganzes
- Enormer Wert für das Finden von Regressionen, speziell in der Zusammenarbeit zwischen Komponenten
- Zumindest einige davon sollten End-to-End sein, also z.B. vom webbasierten UI bis zur Datenbank möglichst Komponenten berühren
- Manchmal Tradeoff zwischen Komplexität und Verlässlichkeit
 - UI Tests sind oft nicht 100% stabil
 - Zu viele *false positives* zerstören die Signalwirkung

Continuous Delivery

- Noch ein Schritt weiter: Jeder Commit landet potenziell in der Produktion
- Braucht:
 - Solide Integrationstests
 - Monitoring der Betriebsdaten (real user data)
 - Strategien zur Risikominimierung (z.B. blue-green-deployments, Rollback)
 - Betrieb der eigenen Anwendung



<https://www.slideshare.net/mikebrittain/principles-and-practices-in-continuous-deployment-at-etsy>

Was tun als on-premise Vendor? Unsere Story

- Es gibt viele exzellente Vorträge und Blogeinträge von SaaS-Firmen über Continuous Delivery
 - z.B. Netflix, Etsy, Github
- Die Workload Automation Produkte von CA sind Standardsoftware, die on premise läuft.
 - Schnell iterieren = Wettbewerbsvorteil
 - Wie kann man diese Feedbackschleifen in unserem Kontext herstellen?
 - Die folgenden Folien kommen von meinem Kollegen Christopher Hejl, der dieses Projekt geleitet hat
 - Ganzer Vortrag: <https://www.youtube.com/watch?v=xVtyrclicDk>
 - Slides: „Automic“ = Firmenname vor Akquisition durch CA

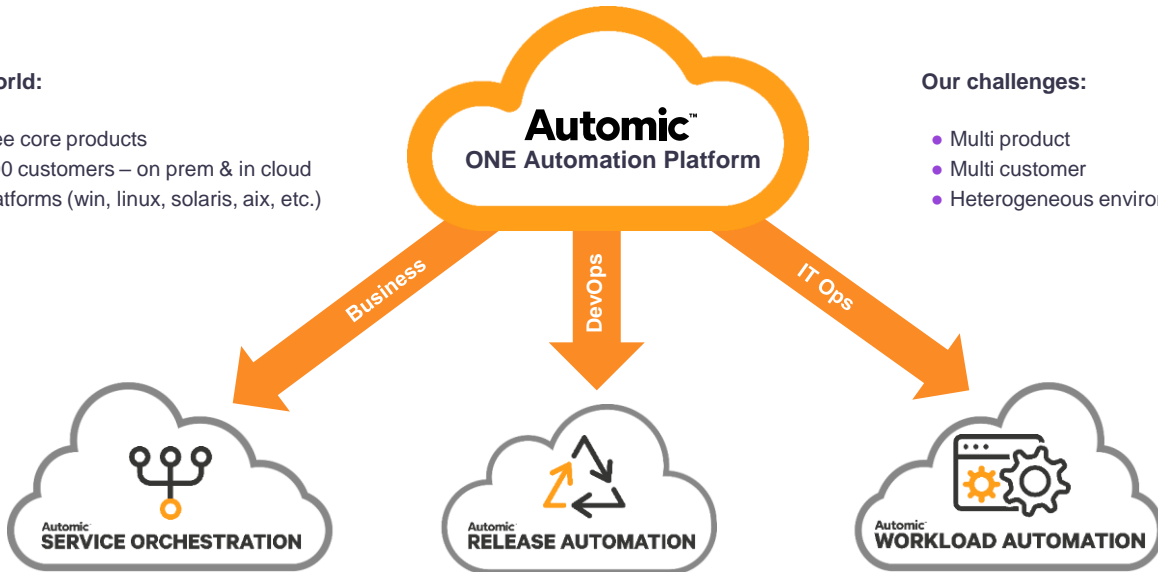
Automic in 2 minutes (in 2017)

Our world:

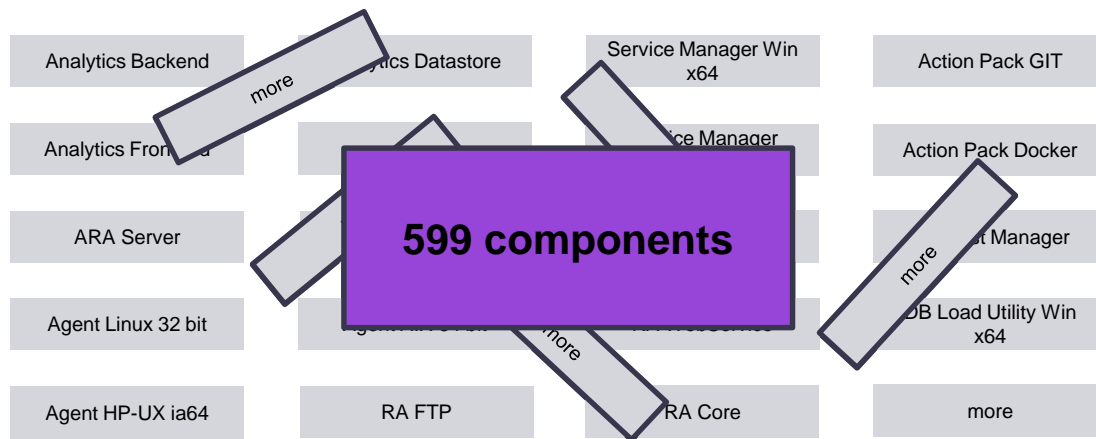
- Three core products
- 2.700 customers – on prem & in cloud
- x platforms (win, linux, solaris, aix, etc.)

Our challenges:

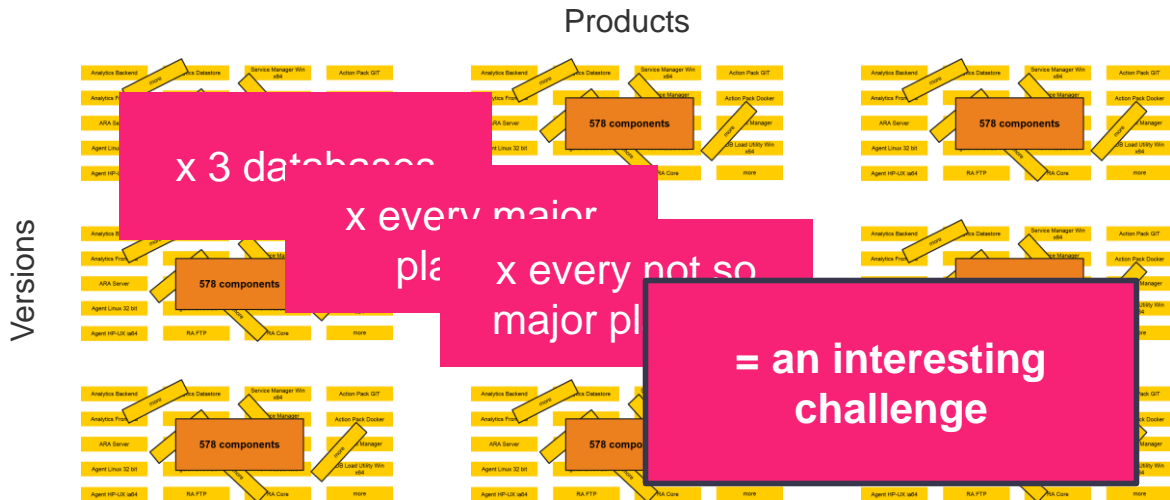
- Multi product
- Multi customer
- Heterogeneous environments



Components and Products



Components and Products

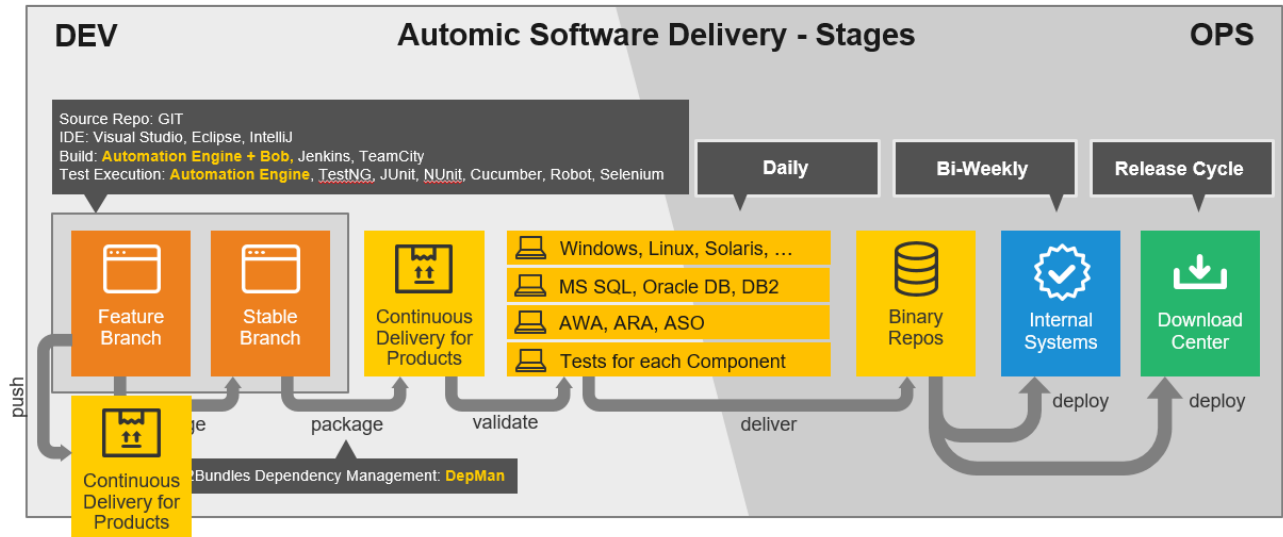


Let's see how others solve this

404

Page not found

How does it work in detail?





Code

Everything starts with a ticket

Branch off

Adapt code

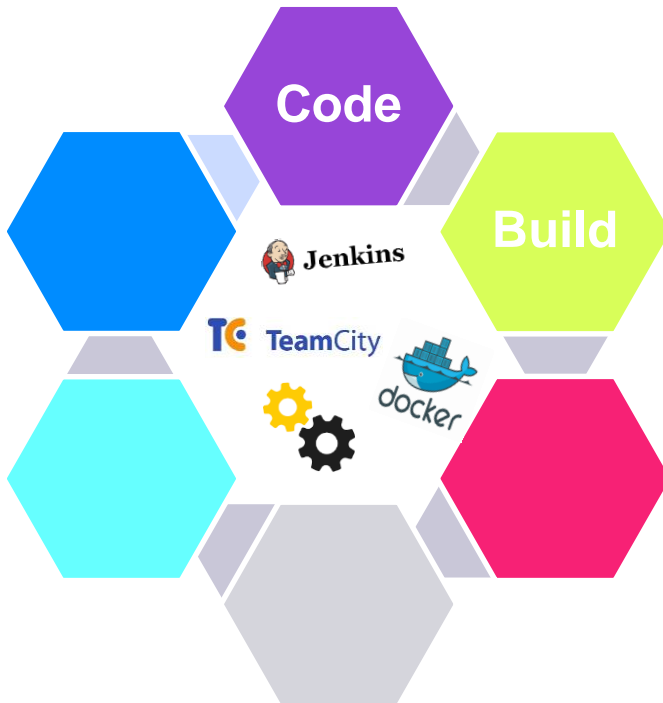
Run local tests

Commit/push

Merge/pull request

All done via

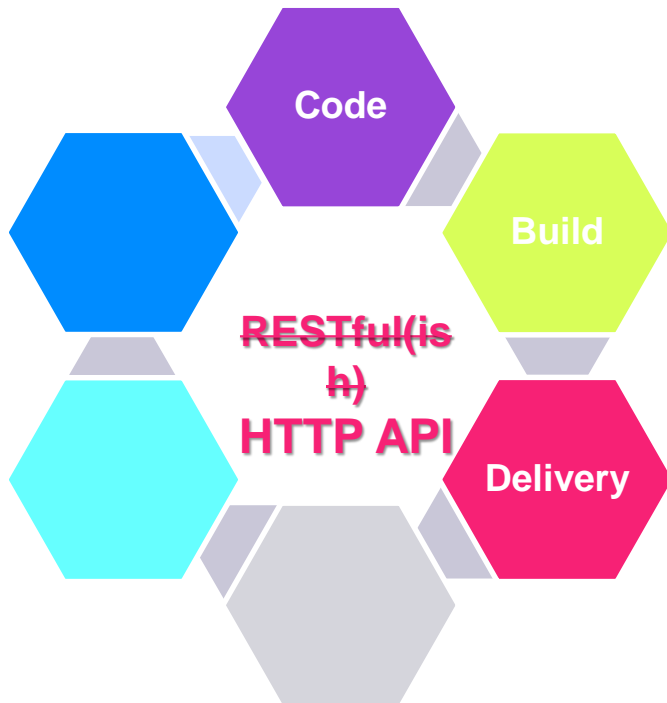
- Ticket-system
- IDE



Build

- Pull code & dependencies
- Run unit tests
- Run other tests
- Build software
- Package software
- Build test image*
- Upload test image

* Docker image which will run tests in a live environment

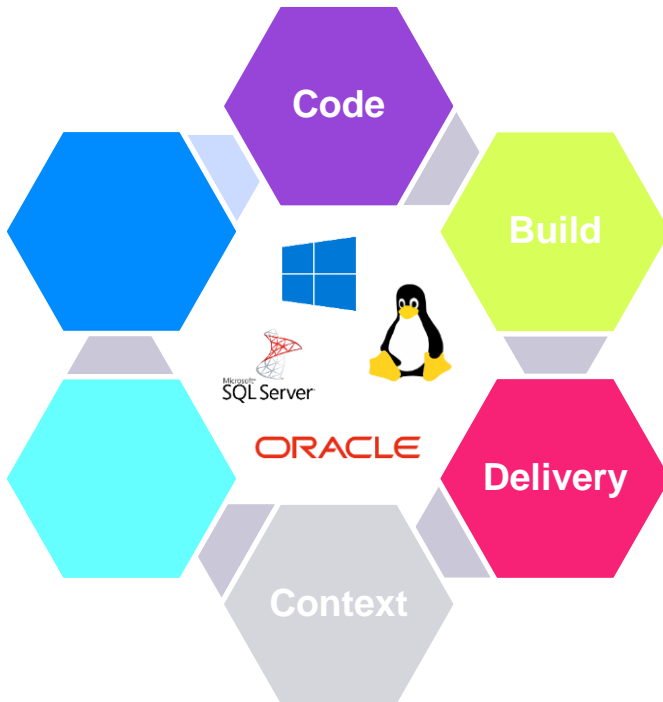


Delivery

HTTP API

- Build artifact
- Test image*
- Repo
- Branch
- Commit
- Build report

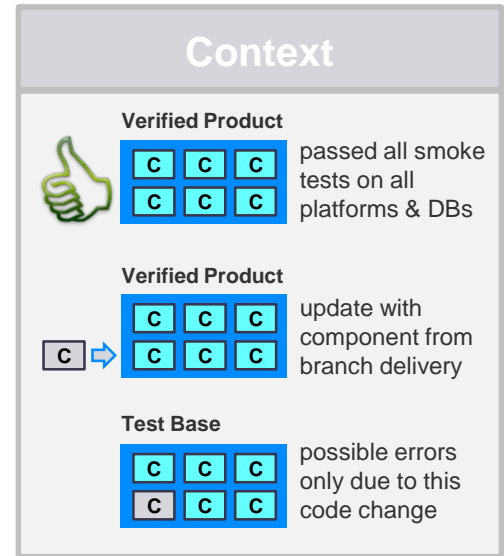
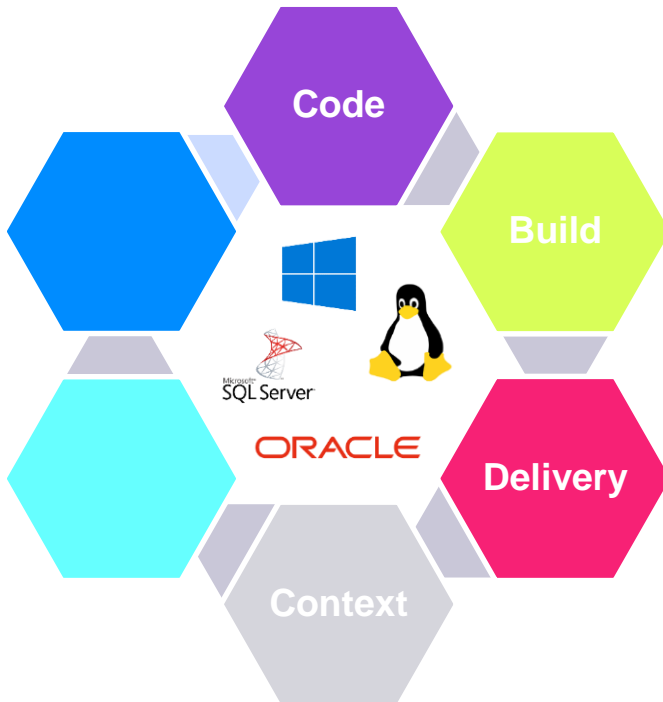
* on the Docker registry

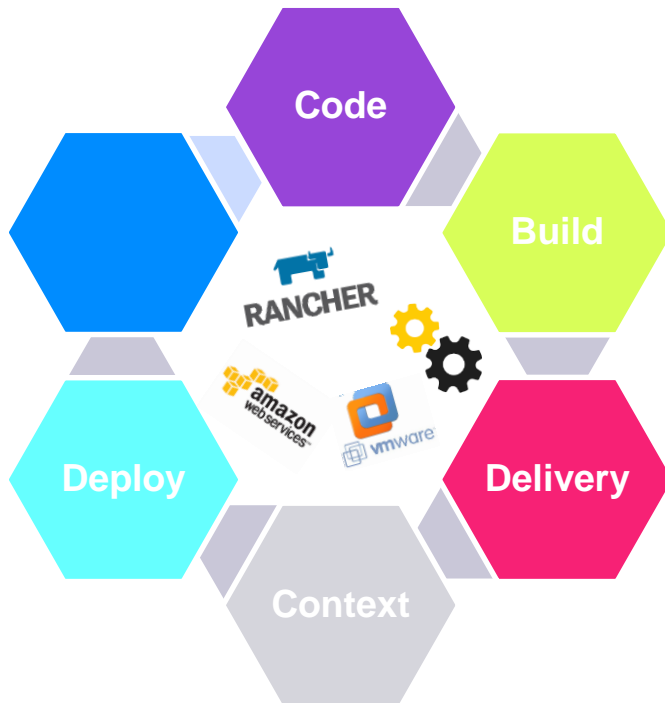


Context

Identify affected product
Map supported OS & DB

- **Specific**
Engineer specified context in ticket, commit message, on demand form
- **All**
Change was on a release branch
- **Random***
Branch is not a release branch and nothing was specified





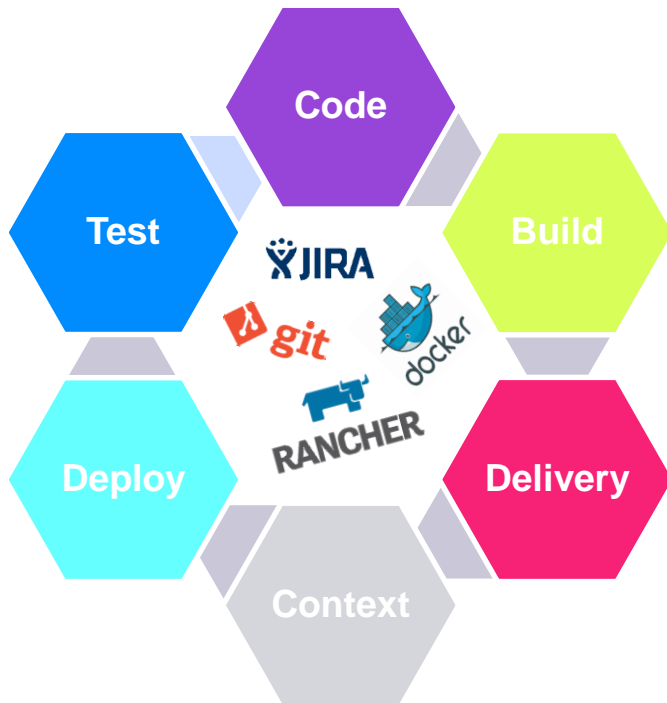
Deploy

Runtime test dependencies
DB, Application Server, Java, AD, ...

Dependent on the software
We execute an automated software
deployment process for our products

De-provision after test
With an optional delay for manual
testing

Private and/or public cloud



Test

Spawn every test image
from every component

Your tests guard your component
from your changes **and** breaking
changes in other components

Unified way to run test

Provided with Docker while
enabling dev teams to define their
own environment

Results shown in Jira
ticket

Parsing via xUnit-style reports
Option to provide additional output

Was heißt das für mich als Entwickler?

- Sicherheit, dass meine Änderung funktioniert
- Ich finde heraus, wenn ich eine Schnittstelle breche, von der ich nichts wusste
- Änderungen über mehrere Komponenten leichter
- Perspektivenwechsel: Gesamtanwendung

Positives Feedback aus der Organisation

Aber: Akzeptanz steht und fällt mit Geschwindigkeit

Feedback für das Produkt

Genauso wichtig wie die anderen Themen aber nicht meine Kernkompetenz, daher nur einige Hinweise:

- Mit Kunden sprechen
 - Beziehungen aufbauen und pflegen
 - Prototypen validieren
 - Aktiv Feedback einholen
 - Kontakte zu Power-Usern pflegen
- Nutzerverhalten messen
 - Usability tests
 - Mit Einverständnis Daten über die Nutzung von live-Systemen sammeln

Aber...

- Unvollständige Liste von Dingen, die man in der Praxis antrifft und tendenziell nicht erfolgreich neu lösen wird (Ergebnis aus Jahrzehnten Forschung & Praxis)
 - Parser
 - Fundamentale Algorithmen
 - Kryptografie
 - Datenbanken
 - Verteilte Systeme
 - Nebenläufigkeit
- Auf Feedback reagieren ist orthogonal zu Planung, Sorgfalt und Wissen - kein Ersatz!
 - Beispiel: TDD kein Allheilmittel

Zusammenfassung

- Wir lernen durch Rückmeldung aus unserer Umgebung
- Viele kleine, rechtzeitige Kurskorrekturen führen oft zum Erfolg
- Wir müssen uns aktiv um Feedback bemühen
 - Auf persönlicher Ebene
 - In unserem Entwicklungsprozess
 - Beim Programmieren (individuell und für das Team)
 - Für das Produkt



Danke!