Datenanalysen in der Softwareentwicklung mit Software Analytics

Hackerkegeln, Nürnberg
15.02.2018

Drucker- und Urheberrechts-freundliche Version inkl. Erläuterungen

Markus Harrer

@feststelltaste feststelltaste.de talk@markusharrer.de

Abstract

Bei unseren altgedienten Anwendungssystemen bekommen wir manchmal das Gefühl, dass irgendetwas grundlegend schief läuft. Das Business lässt sich aber nur mit Zahlen, Daten und Fakten von dringenden Verbesserungsarbeiten überzeugen. Uns Entwicklern fehlt jedoch die Zeit zum Sammeln und Aufbereiten der Indizien hin zu soliden, stichhaltigen Argumenten.

Hier können wir selbst nachhelfen: Jeder Schritt in der Entwicklung oder Verwendung von Software hinterlässt wertvolle, digitale Spuren. Diese Daten können durch geschickte Auswertungen Problemursachen in unserer Software für jeden klar und deutlich aufzeigen. Sind die Probleme und ihre Auswirkungen erst einmal bekannt, können Entwickler und das Business gemeinsam Lösungen mit einem passenden Aufwand-Nutzen-Verhältnis erarbeiten.

In meinem Vortrag stelle ich einen digitalen Notizbuchansatz sowie Werkzeuge für die schnelle Durchführung von nachvollziehbaren Datenanalysen in der Softwareentwicklung vor. Dadurch lassen sich ganz individuell Problemursachen in Softwaresystemen Schritt für Schritt herausarbeiten und explizit darstellen. Ich zeige das Zusammenspiel von Open-Source-Analysewerkzeugen (wie Jupyter, Pandas, jQAssistant, Neo4j und D3) zur Untersuchung von Java-Anwendungen und deren Umgebung (Git, FindBugs, JaCoCo, Profiler, Logfiles etc.). Ich stelle auch einige meiner Auswertungen zu Race-Conditions, Performance-Hotspots, Wissenslücken und wertlosen Codeteilen vor – von den Rohdaten bis zu Visualisierungen komplett automatisiert ausgeführt.

Markus Harrer

Software Development Analyst

@feststelltaste

Blog: feststelltaste.de

Clean Code, Agile, Softwaresanierung, Softwarerückbau

Wer bin ich

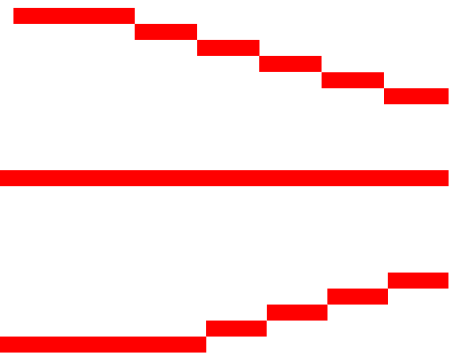
Softwareentwickler: Java-Entwickler mit Fokus auf Clean Code und agiler Softwareentwicklung sowie Vorliebe für Softwaresanierung und Softwarerückbau.

Software Development Analyst: Analyst der Softwareentwicklung, um Problemursachen sichtbar zu machen und Lösungsoptionen zu erarbeiten.

Freiberuflicher Trainer und Consultant: Angebot von Workshops und Trainings für Datenanalysen in der Softwareentwicklung. Beratung zum strategischen Refactoring von Legacy Systemen.

Agenda

- Warum?
- Wie?
- Was?



Datenanalysen in der Software-entwicklung

Inhalte

Wir sehen uns an,

- WARUM Datenanalysen in der Softwareentwicklung notwendig sind
- WIE Analysen im Softwarebereich effizient und handlungsorientiert umgesetzt werden können
- WAS sich hier bereits konkret umsetzen lässt

WARUM?

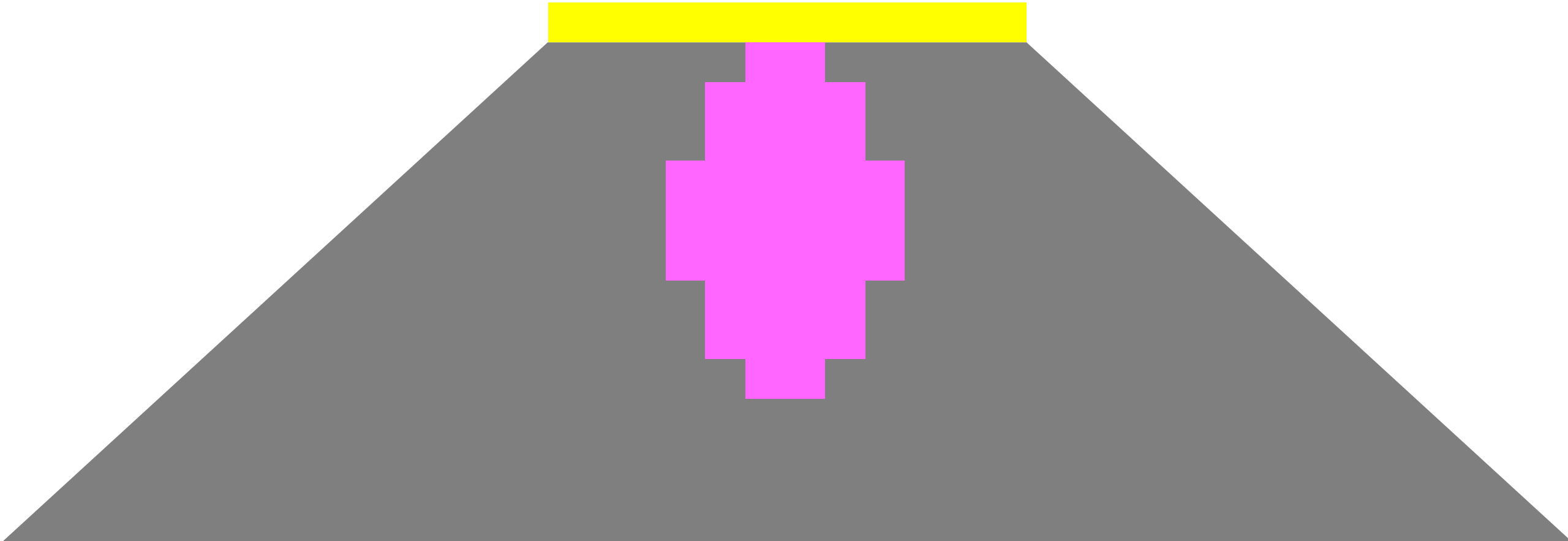
Warum?

Als Softwareentwickler werden wir mit einer ganz eigenartigen Mischung von zweierlei Problemkategorien konfrontiert, die sich überhaupt nicht lösen lassen.

Wir können sie nur kennenlernen und selbst lernen, damit angemessen umzugehen.

Wo wir auf diese Probleme stoßen, sehen wir uns jetzt an.

Management



Einführendes Beispiel

Stellen wir uns dazu ein Stück unserer Software vor – den für das Management und dem Business sichtbaren Teil der Anwendung. Diese kann oft als „goldene Oberfläche“ gesehen werden. Hier ist alles picobello!

Warum sollte wir es denn Nicht-Technikern dann übel nehmen, dass sie „unter der Haube“ der Anwendung nicht auch denken, dass sich hier ein „schöner Diamant“ befindet?

HAPPINESS

Happiness all around the World

Also ist doch alles super in der Softwareentwicklung! Softwareentwickler müssen zu den glücklichsten Menschen auf der Welt zählen.

Oder?

Abends in der Kneipe...

Aber...

Wer wirklich wissen will, wie es mit der Software unter der Haube aussieht, kann sich einfach einmal auf ein paar Bier mit den Softwareentwicklern treffen. Denn viele Unterhaltungen zwischen Entwicklern an der Bar (oder in der Kaffeeküche) laufen vielleicht ein bisschen „anders“ ab.

Sarcasm Island

The home of cynicism

Entwickler am Abend

Machen wir doch einfach selbst ein Bild und gehen in eine Kneipe auf der schönen Insel „Sarcasm Island – The home of cynicism“. Es finden sich bestimmt ein paar Entwickler, mit denen wir Quatschen können.

Unsere Unterhaltungen könnten hier wie folgende aussehen.

Symptombehebung

Wir haben jetzt eine
7-Schichtenarchitektur!

Wie ist es denn dazu gekommen?

Jedes Jahr machen wir eine neue
Schicht, um den
Mist vom letzten Jahr zu
überdecken!

Entwickler: Symptom-Behebungen

Als Entwickler im täglichen Hamsterrad ist man leider Gefangener von Entscheidungen aus der Vergangenheit, welche damals wohl zurecht richtig erschienen, aber sich heutzutage falsifiziert haben. Oft wird daher nur immer der Workaround auf dem Workaround gesetzt und nie wirklich die eigentlichen Problemursachen aus den üblichen „Argumenten“ „keine Zeit“, „sind eh bald fertig“ oder „bin eh bald weg“ adressiert.

So sind heutzutage viele existierenden „Schichtenarchitekturen“ in Softwaresystemen zu „Make-Up-Architekturen“ geworden: Defizite werden mittels Schichten überdeckt, um ein makelloses Gesamtbild nach außen hin erscheinen zu lassen.

Intransparenz

Wir hatten überall in der Software Bugs!

Wie seid ihr damit umgegangen?

Unser Vertrieb hat den Kunden abnehmen lassen. Nun sind alle Bugfixes teure Erweiterungen!

Produktmanagement: Intransparenz

Die Problemverdrängung macht einen kreativen Umgang mit der Wahrheit notwendig. So werden Kunden oft Bugs als kostenpflichtige Erweiterungen verkauft oder Software bewusst inkl. Vendor-Lock-In konzipiert, welcher Kunden den Wechsel auf Alternativen verwehrt oder nur mit unverhältnismäßig hohen Aufwänden möglich macht.

Software als immaterielles, also nicht sichtbares Gut ist diesen Bestrebungen schutzlos ausgeliefert. Entwickler können nicht sofort nachweisen, welche Risiken Nicht-Techniker mit den von ihnen vorgenommenen (oder unterlassenen) Entscheidungen bei der inneren Softwarequalität eines Anwendungssystems bewirken.

Wahrnehmungsdiskrepanz

Gestern ist unser jahrelang entwickeltes System in Produktion abgeraucht!

Schrecklich! Was ist passiert?

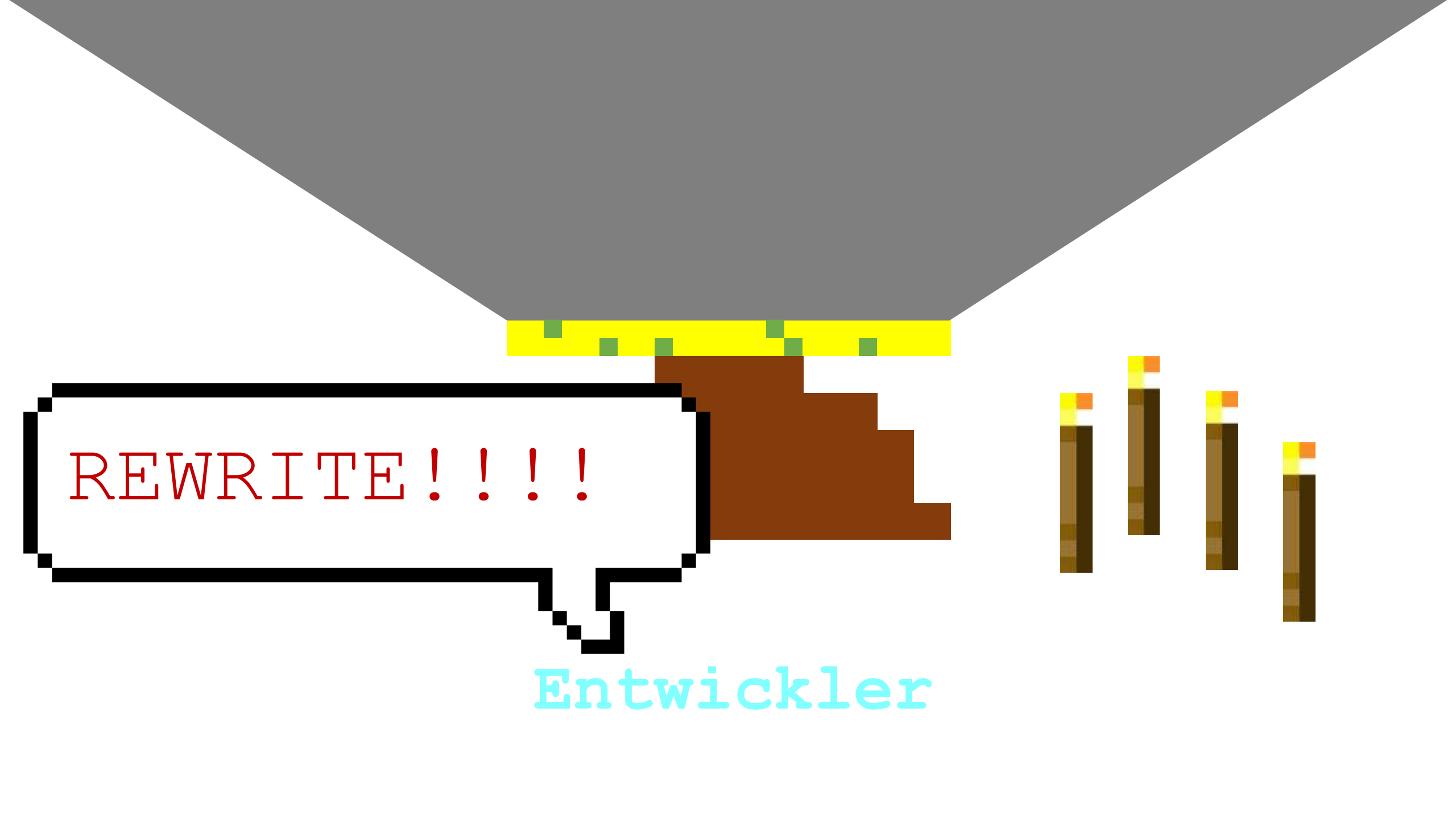
Nichts! Niemand verwendet es!

Management: Wahrnehmungsdiskrepanz

Die Realität sieht im Gegensatz zum vermittelten Bild an das obere Management oft komplett anders aus. Es bedarf daher immer eines disruptiven Ereignisses, um Wunsch und Wirklichkeit wieder in den Einklang zu bringen. Zu oft ist dies dann aber der Tod des Altsystems oder der Firma selbst.

Im Grunde dürfen wir uns hier nicht darüber wundern: Schwer kommunizierbare, technische Defizite stehen oft Hochglanz-Prospekten gegenüber. Es ist nicht schwer zu erraten, welche Informationen bei der Unternehmensführung gesehen werden: Statt Hiobsbotschaften kommen hier in Power-Point gewickelte Happiness-Häppchen zur leichten Verdauung an.

Das Problem: Oft ist das den Beteiligten gar nicht bewusst!


REWRITE!!!!
Entwickler

Die andere Seite der Medaille

Entwickler haben einen anderen Blick auf die Software: Sie sehen den Code. Zuerst machen sich die Defizite der Software durch geringere Produktivität der Entwickler bemerkbar. Danach kommen Qualitätsdefizite auch an der Oberfläche oder bei der Bedienung zum Vorschein. Nach einiger Zeit ist der gemeinsame Druck der Entwickler und auch der enttäuschten Kunden auf das Management so hoch, dass die Neuschreibung der Software bewilligt wird.

Das Spiel wiederholt sich alle 3-5 Jahre.

Das ist der einzige iterativ durchgeführte Prozess der in der Softwareentwicklung, der wirklich immer und immer wieder zuverlässig funktioniert!

FAIL!!!

Neuschreibung ist keine Problemlösungsstrategie!

Das ist ein sehr unbefriedigender Zustand und muss so nicht sein!

„Die Definition des Wahnsinns ist, immer dasselbe zu tun und ein anderes Ergebnis zu erwarten.“

– Albert Einstein

Warum ist Softwareentwicklung

immer noch so

wahnsinnig?

Software Development

is the act of transforming

Knowledge

into

Code

If you want to fix this

you have to address that!

Grundproblem 1: Falscher Umgang mit dem Ding „Wissen“

Als Entwickler beschäftigen wir uns zu stark mit der Softwareentwicklung selbst und unserem Code. Wir haben tolle Tools und famose Frameworks geschaffen. Aber zu selten stellen wir uns die Frage, mit was wir überhaupt arbeiten.

Softwareentwicklung ist die Überführung von Wissen in Code. Und „Wissen“ selbst hat viele Facetten, welche den Umgang damit schwierig gestalten. Die philosophische Disziplin der Erkenntnistheorie fragt hier: „Was ist Wissen?“ und „Kann es so etwas wie Wissen überhaupt geben?“

Das Ergebnis ist, dass es viele unlösbare Dilemmata gibt, wie etwa die Vergänglichkeit und Kontextabhängigkeit von Wissen oder die Nichtformalisierbarkeit bestimmter Phänomene in der Realität. Oder auch die Gefahren, die bestehen, wenn vorhandenes Wissen nicht kritisch hinterfragt wird (oder werden darf). Dennoch kennen die wenigsten Entwickler diese Einschränkungen. Jeglicher Versuch der Implementierung von Wissen in Code oder auch das Schaffen von Werkzeugen zum Umgang mit Quellcode ist ohne Beachtung dieser fundamentalen Dilemmata aber ein gefährliches Spiel mit dem Feuer.

Zum Glück gibt es aus anderen Disziplinen Methoden und Vorgehen, welche zumindest die Defizite im Umfang mit dem Wissen akzeptieren und mit entsprechenden Maßnahmen behandeln.

Behavioral *ECONOMICS*™

Grundproblem 2: Behavioral Economics

Das zweite Bündel an Problemen wird uns von der Verhaltensökonomik beschert. Diese zweifelt sehr stark an dem Bild der rational handelnden Menschheit. Zwar sind wir alle moderne Menschen, doch in einigen Handlungen unterscheiden wir uns nicht von unseren steinzeitlichen Vorfahren oder den heutigen Primaten. Besonders bei Entscheidungssituationen unter Stress sind wir auf Gedeih und Verderb unseren urzeitlichen Instinkten ausgeliefert. Zwar haben die hier eintretenden „kognitive Verzerrungen“, welche z. B. durch die Arbeit in Gruppen, bei Fluchtgedanken oder Risikobetrachtungen ausgelöst werden, heute andere Effekte als im Kampf gegen den Säbelzahntiger, dennoch beeinflussen sie unser Verhalten im Umgang mit Entscheidungen fundamental.

Auch lassen sich die Auswirkungen hier wiederum nur begrenzen, wenn sie bekannt sind und vor einer Entscheidungsfindung aktiv adressiert werden. Entsprechende Ansätze zur Verbesserung von Softwaresystemen müssen dies ebenfalls tun!

Management

RISIKO

MAUER DER IGNORANZ

SICHTBARKEIT

Controller

Datenanalysen durch Controller

Zum Glück gibt es im Bereich der Wirtschaftswissenschaften bereits einige Maßnahmen, um die Auswirkungen unserer kognitiven Defizite wirksam zu begegnen.

Sehen wir hier die Zusammenarbeit zwischen dem Management und dem Controlling an. Das Management als Unternehmensführung muss als unternehmerisch handelnder Akteur Risiken eingehen dürfen. Konkret heißt das z. B. die Erschließung neuer, unsicherer Märkte bevor es die Konkurrenz tut. Das Controlling als Unternehmensteuerung versucht durch die wirtschaftliche Betrachtung der Managementhandlungen, die Auswirkungen von Risiken sichtbar zu machen.

Beide Welten werden nun jedoch von einer Art „Mauer der Ignoranz“ getrennt, was stellvertretend für unsere kognitiven Defizite stehen soll.

Management

RISIKO

DATENANALYSEN

SICHTBARKEIT

Controller

Datenanalysen für Sachlichkeit

Es braucht hier ein Kommunikationsmittel, welches beide Seiten miteinander verbinden kann. Im strategischen Controlling ist dies die Datenanalyse. Hier werden die rohen Geschäftsdaten so aufbereitet, dass sie dem Management in seiner Risikoabwägung unterstützen können.

Das oft trügerische Bauchgefühl wird hier durch harte Fakten ersetzt. Zahlen stellen die Auswirkungen des unternehmerischen Handelns neutral darf.

Management

RISIKO

DATENANALYSEN

SICHTBARKEIT

Entwickler

Datenanalysen durch Entwickler!?

Um die eingegangenen Risiken in der Softwareentwicklung nun entsprechend handhaben zu können, sollten auch wir Entwickler in die Rolle des Controllers schlüpfen. Auch wir können mittels Analysen unserer eigenen Daten frühzeitig Risiken identifizieren und managementgerecht kommunizieren.

Wie das umgesetzt werden kann, sehen wir uns im zweiten Teil genauer an.

Wie?

Datenanalysen – Alter Hut?

Es gibt einen ganzen Bereich in der akademischen Software Engineering Zunft, welcher sich die Nutzung von Softwaredaten zur besseren Steuerung der Softwareentwicklung auf die Fahnen geschrieben hat: Empirical Software Engineering.

Bei Empirical Software Engineering haben aufgeblähte Metrikgebilde oder wackelige Argumentationsketten keine Chance. Forschungsarbeiten im Empirical Software Engineering werden rigoros validiert, repliziert und auch gegenseitig zerfetzt. Es ist ein hartes wissenschaftliches Forschungsfeld und bringt jedoch sehr wertvolle Vorgehen und Techniken hervor.

Eine konkrete Ausgestaltung der empirischen Softwareentwicklung – Analysen auf Basis von Softwaredaten – lernen wir nun genauer kennen. Und wir sehen uns auch an, wir sie als Entwickler selbst nutzen können.

Software Analytics – Eine Definition

"Software analytics is

analytics on **software data** for
managers and **software engineers**

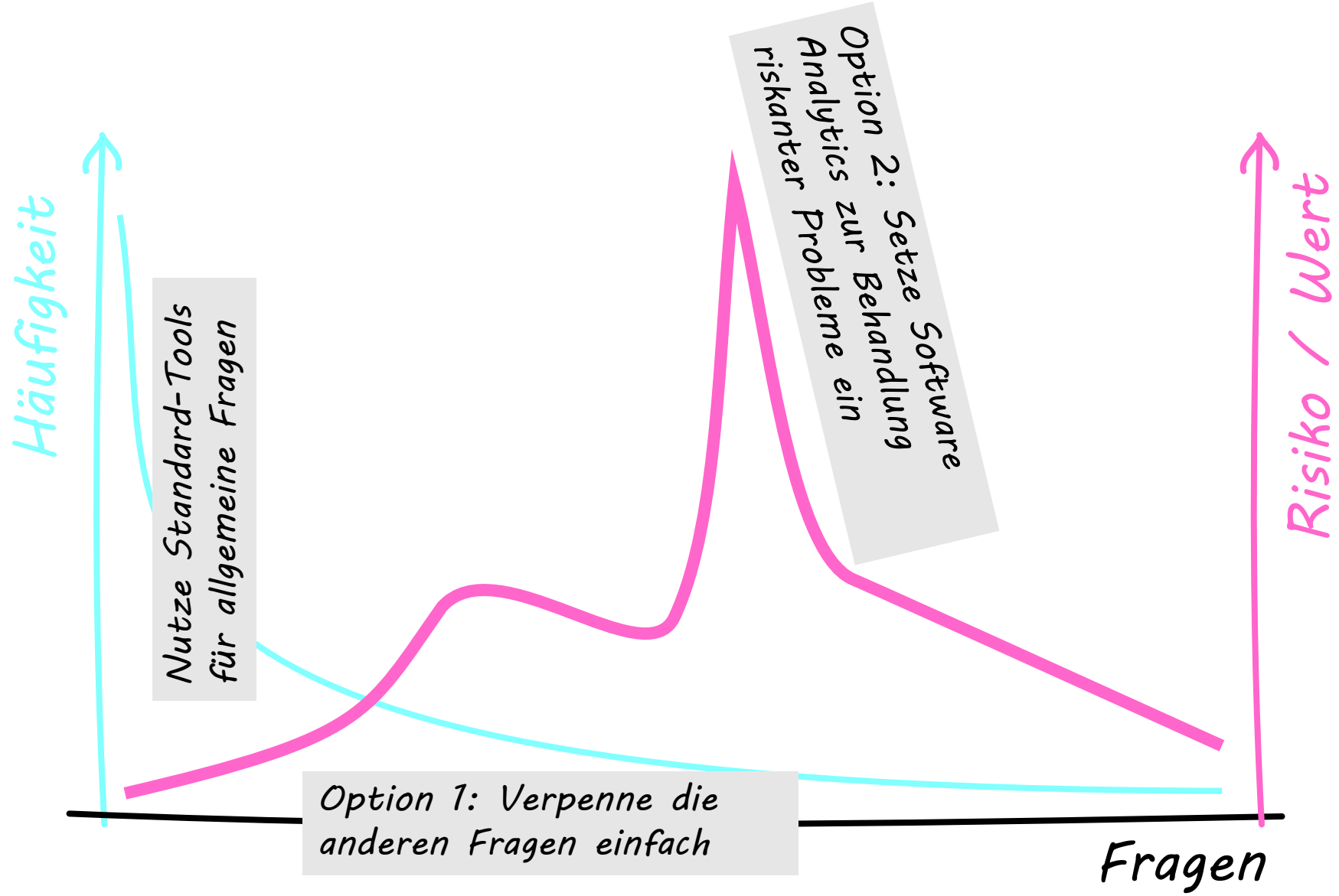
with the aim of **empowering software development individuals and teams**

to gain and share **insight** from their data
to make better **decisions**."

Software Analytics...

- ...führt Analysen auf Basis der Daten durch, die während der Entwicklung oder dem Betrieb von Software entstehen
- ...hat als Zielgruppe sowohl die Softwareentwickler als auch das Management (ein sehr essentieller Punkt)
- ...möchte alle an der Entwicklung Beteiligten dazu befähigen, neue Einsichten aus den Softwaredaten zu gewinnen
- ...um bessere Entscheidungen für die Softwareentwicklung treffen zu können

Software Analytics fokussiert sich auf wichtige Fragen



Wozu Software Analytics?

Daten aus der Softwareentwicklung auswerten? Das kommt uns irgendwie bekannt vor. Klar haben wir bereits Tools im Einsatz, welche uns die am häufigsten gestellten Fragen beantworten können. Klassische, statische Code-Analysewerkzeuge weisen uns auf typische Programmierfehler hin – und das ist gut. Das schlechte daran ist aber, dass die Häufigkeit einer Frage nicht mit dem Risiko dieser Frage korreliert: Die höchsten Risiken könnten bei einer nicht sehr häufig gestellten Frage auftreten.

Nun hat man zwei Möglichkeiten, darauf zu reagieren: Einerseits kann man so tun als ob es dieses Problem gar nicht gibt. Oder man nutzt dedizierte Analysen, um diese Art von Problemen zu identifizieren und auch zu lösen. Genau hier sehe ich den Einsatzbereich von Software Analytics.

Legacy Code

An Adventure
65 Years In The Making

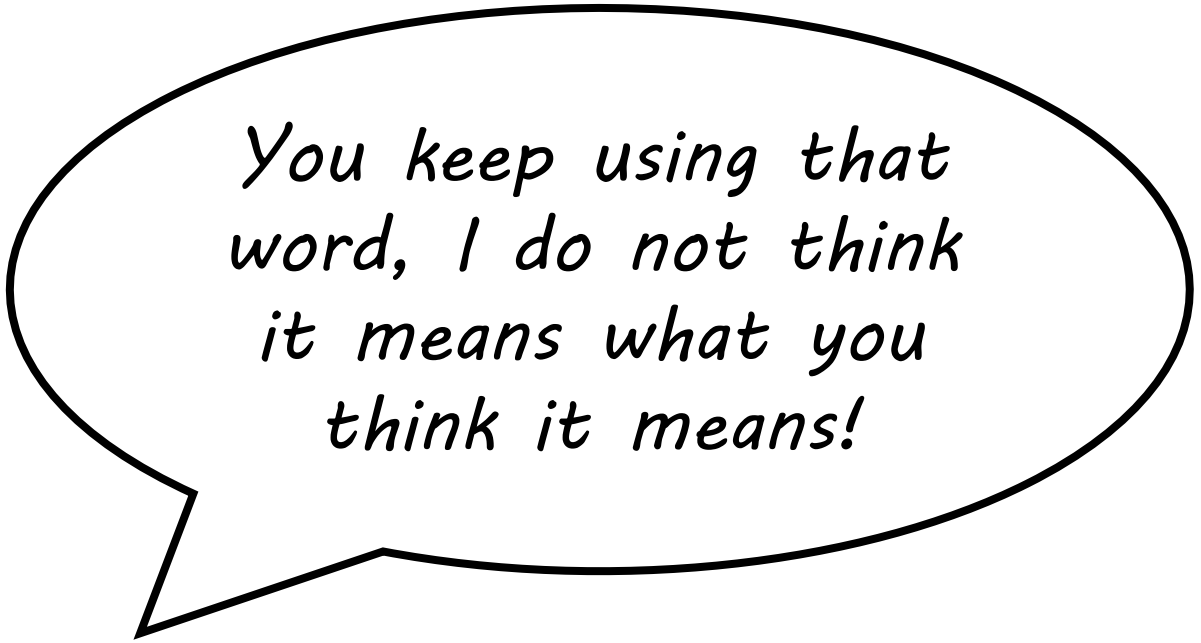
FIND LINKS BETWEEN COMPONENTS
MINING PERFORMANCE DATA
USING EXECUTION TRACES TO LEARN FROM PROGRAM USAGE PATTERNS
PREDICT WHERE CODE WILL FAIL
CLASSIFY CHANGES AS CLEAN OR BUGGY
USING VISUALIZATION TO SUPPORT PROGRAM COMPREHENSION

Anwendungsbeispiele

Das Anwendungsfeld von Software Analytics ist breit gestreut:

- Identifikation von Zusammenhängen zwischen beliebigen Belangen aus der Softwarewelt
- Herausfinden von Performance-Problemen in Anwendungen
- Von Ausführungsaufzeichnungen auf die normale Nutzung von Programmen schließen
- Vorhersagen treffen, welcher Code besonders fehleranfällig ist
- Änderungen nach ihrer Fehlerträchtigkeit einordnen
- Mit Visualisierungen das Verständnis über die Zusammenhänge in der eigenen Software darstellen

THE SCIENTIFIC METHOD



Die wissenschaftliche Methode

Analytics – was heißt das jetzt eigentlich für uns Softwareentwickler?

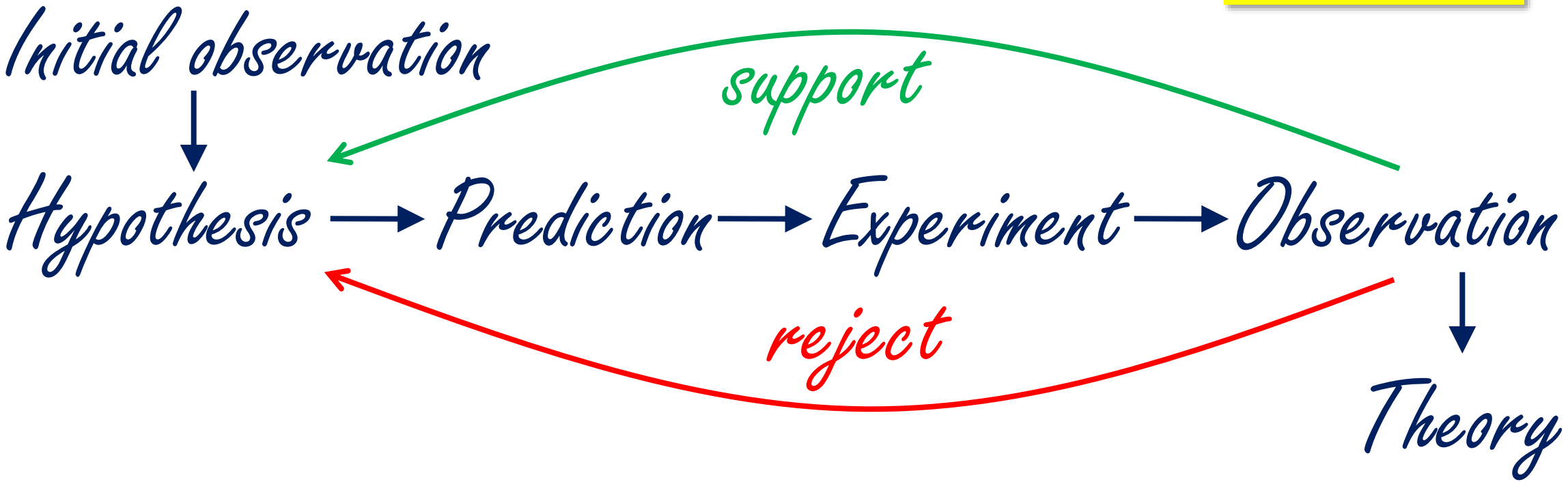
Für mich ist die wichtigste Neuerung, dass wir uns nun nicht mehr – wie bisher – vom Analyseergebnis eines Tools zu den eigentlichen Problemen bewegen, sondern von einem konkreten Defizit der eigenen Anwendung aus zur Problemursache. Softwaredaten sind nur noch unterstützendes Mittel zum Zweck, um zu nachweisen zu können, dass wir in der eigenen Anwendung vielleicht ein schwerwiegendes, chronisches Defizit vorliegen haben, welches evtl. den Live-Gang unseres Projekts gefährdet oder das System in Produktion abstürzen lässt.

Und um sich von einer konkreten Fragestellung hin zu einer gefestigten Theorie hin zu bewegen, haben wir ja bereits eine Methode entwickelt, die genau das macht: Das wissenschaftliche Arbeiten. Genau hier sind bereits die Probleme adressiert, die im Umgang mit Wissen entstehen können. Und wir können auch die erprobte Vorgehensweise für unsere Analysen wiederverwenden.

The Scientific Method

data

automation



Vorgehensweise wissenschaftliche Methode

Die wissenschaftliche Methode geht von einer initialen Beobachtung aus wie z. B. einem aufgetretenen Bug oder einer „komischen“ Codestelle in unserer Software. Der nächste Schritt ist dann aber nicht (wie so oft bei uns Entwicklern) die unmittelbare Lösung (aka Workaround oder Quickfix).

Die wissenschaftliche Methode geht hier weiter: Zuerst wird über die Erstellung einer Hypothese versucht, die Problemstellung genau zu verstehen. Hier wird geklärt, ob wir wirklich eine Problemursache gefunden haben oder ob wir nur ein Symptom eines tiefergehenden Problems sehen. Im nächsten Schritt versuchen wir, das Problem zu „formalisieren“, also in eine Form zu bringen, damit wir es an anderen Stellen im Code vielleicht vorhersagen können. Wir bauen ein Experiment oder eine Analyse, welche uns unsere Hypothese bestätigt. Die hier gewonnen Ergebnisse versuchen wir zu falsifizieren. Gelingt uns das nicht, haben wir wirklich ein tieferliegendes, chronisches Problem identifiziert, welches wir nun dazu verwenden können, entsprechende Problemstellen zu finden. Nehmen wir zu dieser Methode noch Daten dazu und automatisieren wir das Vorgehen, dann kommen wir in den Bereich, in dem ich eine mögliche, pragmatische Umsetzung von Software Analytics sehe.

REPRODUCIBLE DATA SCIENCE

= ONE WAY TO IMPLEMENT
SOFTWARE ANALYTICS

Reproducible Data Science

Automatisierte, datengetriebene und nachvollziehbare Analysen von Softwaredaten – offen, für jeden einsehbar, wiederholbar und verständlich aufbereitet von den Rohdaten bis zu managementtauglichen Visualisierungen.

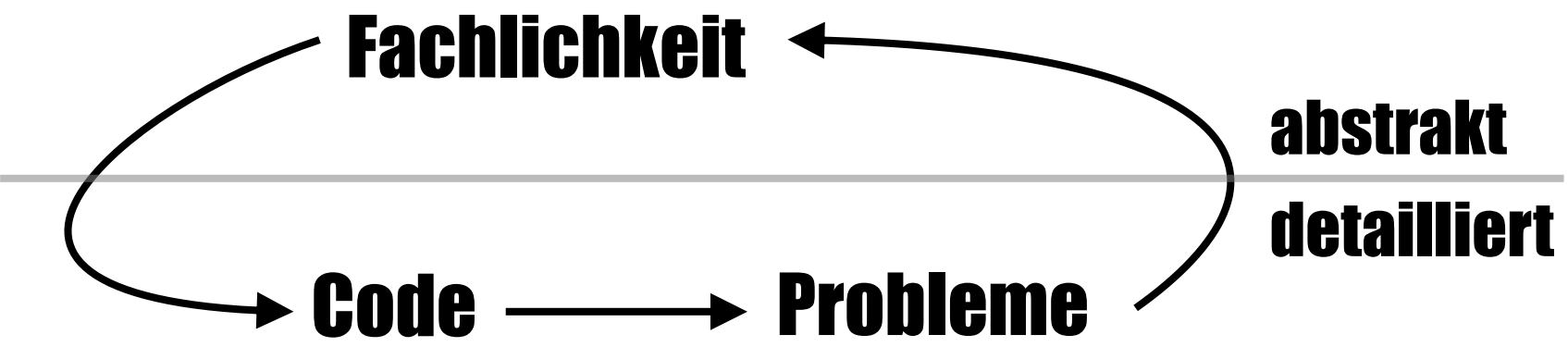
Die Methoden und Vorgehensweisen von Reproducible Data Science sind für mich die Leitplanken zur pragmatischen Umsetzung von Software Analytics.

Warum gerade jetzt?

Code verschmilzt mit Fachlichkeit

Data Science bringt Daten-analysen zu Entwicklern

Werkzeuge bilden und vernetzen verschiedene Perspektiven



Probleme können mit fachlichen Konzepten verbunden werden!

Warum jetzt damit anfangen?

Es gibt drei wesentliche Punkte, jetzt in das Thema „Software Analytics“ einzusteigen:

- Durch Domain-Driven-Design und immer feineren fachlichen Nuancen in unseren Anwendungen rücken die fachlichen Belange einer Software noch stärker in den Vordergrund als bisher. Die Fachsprache ist nun direkt im Code
- Die Themen „Big Data“ und „Data Science“ sind nun nicht mehr nur reine Buzzwords, sondern drängen in den letzten Jahren auch immer mehr zu den Entwicklern vor. Entwickler lernen zwangsläufig immer mehr Datenanalysewerkzeuge kennen – als Entwickler oder als Anwender selbst. Dieses Wissen können Entwickler nun nutzen, um damit eigene, ganz individuelle Probleme in der Software aufzudecken
- Doch der entscheidende Punkt für mich ist, dass es nun Werkzeuge gibt, die in der Lage sind, aus den sehr feingranularen, detaillierten Elementen unserer Softwaresysteme abstrakte Konzepte zu bilden. Somit können wir nun verschiedene Perspektiven auf unsere Softwaresysteme für ganz spezifische Anwendungsfälle entwickeln.

Alles zusammen bedeutet, dass wir nun Probleme im Code mit fachlichen Konzepten verbinden können. Somit können wir Entwickler nun unsere ganz Problemen im Code für Nicht-Techniker sichtbar machen und gemeinsam angehen. Damit schaffen wir eine Feedback-Schleife für eingegangene Risiken, die so noch nicht vorhanden war.

Tool Time

Werkzeugkette

Sehen wir uns an, wie das technisch funktionieren kann.

The Notebook

- Kontext dokumentiert
- Ideen, Daten, Annahmen und Vereinfachungen aufgeführt
- Berechnungen verständlich dargelegt
- Zusammenfassungen erklärt

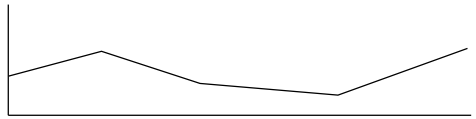
- Komplett automatisiert

Problem

Context

Idea

Analysis



Conclusion

Der digitale Notebookansatz

Wenn wir uns eine wissenschaftliche Abhandlung ansehen, folgt diese meist einer gewissen Struktur, um von der Problemstellung zu einem Ergebnis zu kommen.

Diese Struktur können wir auch in einem digitalen Notebook umsetzen. Ein Notebook bietet so eine Plattform für offene Analysen. Es verbindet Code mit den Daten und zeigt zugleich jedes Ergebnis einer Berechnung an. Die Ergebnisdarstellung ist entweder ein einfacher Text, eine Tabelle oder eine Visualisierung.

Da das Notebook digital ist, läuft auch die Ausführung einer „programmierten“ Analyse automatisiert. Da nur die Rohdaten als Eingabemöglichkeit zur Verfügung stehen, ist jeder getätigter Schritt bis zur Schlussfolgerung nachvollziehbar. Dies macht Analysen zum einen leicht wiederholbar. Zum anderen aber werden vorgenommene Auswertung so gut wie nicht angreifbar. Jeder kann verfolgen, vorher das Ergebnis stammt. Evtl. Vereinfachungen, Fehler oder Manipulationen sind sofort ersichtlich und können bei der Ergebnisdiskussion angesprochen werden.

jupyter Production Coverage Demo Notebook Last Checkpoint: 2 minutes ago (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help

Trusted Python 3

Markdown

Context

John Doe remarked in #AP1432 that there may be too much code in our application that isn't used at all. Before migrating the application to the new platform, we have to analyze which parts of the system are still in use and which are not.

Idea

To understand how much code isn't used, we recorded the executed code in production with the coverage tool JaCoCo. The measurement took place between 21st Oct 2017 and 27st Oct 2017. The results were exported into a CSV file using the JaCoCo command line tool with the following command:

```
java -jar jacococli.jar report "C:\Temp\jacoco.exec" --classfiles \
C:\dev\repos\buschmais-spring-petclinic\target\classes --csv jacoco.csv
```

The CSV file contains all lines of code that were passed through during the measurement's time span. We just take the relevant data and add an additional `LINES` column to be able to calculate the ratio between covered and missed lines later on.

In [1]:

```
import pandas as pd
coverage = pd.read_csv("../input/spring-petclinic/jacoco.csv")
coverage = coverage[['PACKAGE', 'CLASS', 'LINE_COVERED' ,'LINE_MISSED']]
coverage['LINES'] = coverage.LINE_COVERED + coverage.LINE_MISSED
coverage.head(1)
```

Out[1]:

	PACKAGE	CLASS	LINE_COVERED	LINE_MISSED	LINES
0	org.springframework.samples.petclinic	PetclinicInitializer	24	0	24

Analysis

It was stated that whole packages wouldn't be needed anymore and that they could be safely removed. Therefore, we sum up the coverage data per class for each package and calculate the coverage ratio for each package.

In [2]:

```
grouped_by_packages = coverage.groupby("PACKAGE").sum()
grouped_by_packages['RATIO'] = grouped_by_packages.LINE_COVERED / grouped_by_packages.LINES
grouped_by_packages = grouped_by_packages.sort_values(by='RATIO')
```

Beispiel Notebook „Production Coverage“ 1/2

Diese Auswertung ist eine Analyse um festzustellen, welche Code-Teile während des Produktiveinsatzes einer Software verwendet werden und welche nicht.

Die Struktur folgt dem Aufbau einer wissenschaftlichen Abhandlung:

- Abschnitt „Kontext“ schildern die Herkunft des Symptoms sowie die evtl. Problemursache, welche untersucht werden soll.
- Abschnitt „Idee“ führt die Möglichkeiten auf, die das Problem auf Basis vorhandener oder noch zu gewinnenden Daten aufzeigen könnten und skizziert erste Analyseideen sowie erste Datentransformationen.

Analysis

It was stated that whole packages wouldn't be needed anymore and that they could be safely removed. Therefore, we sum up the coverage data per class for each package and calculate the coverage ratio for each package.

In [2]:
```
grouped_by_packages = coverage.groupby("PACKAGE").sum()
grouped_by_packages['RATIO'] = grouped_by_packages.LINE_COVERED / grouped_by_packages.LINES
grouped_by_packages = grouped_by_packages.sort_values(by='RATIO')
grouped_by_packages
```

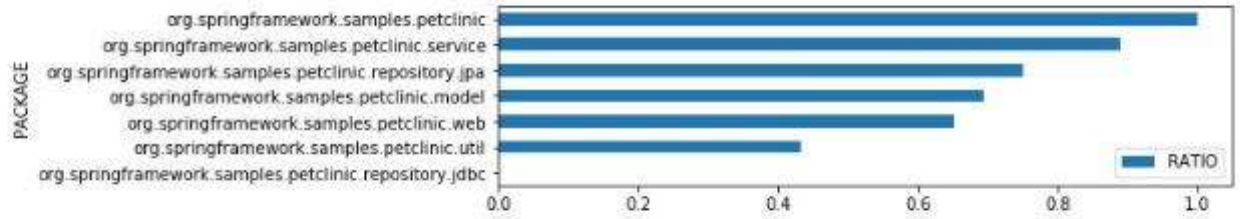
Out[2]:

PACKAGE	LINE_COVERED	LINE_MISSED	LINES	RATIO
org.springframework.samples.petclinic.repository.jdbc	0	152	152	0.000000
org.springframework.samples.petclinic.util	13	17	30	0.433333
org.springframework.samples.petclinic.web	75	40	115	0.652174
org.springframework.samples.petclinic.model	75	33	108	0.694444
org.springframework.samples.petclinic.repository.jpa	21	7	28	0.750000
org.springframework.samples.petclinic.service	16	2	18	0.888889
org.springframework.samples.petclinic	24	0	24	1.000000

We plot the data for the coverage ratio to get a brief overview of the result.

In [3]:
```
%matplotlib inline
grouped_by_packages[['RATIO']].plot(kind="barh", figsize=(8,2))
```

Out[3]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1874cdde9e8>
```



Conclusion

The JDBC package `org.springframework.samples.petclinic.repository.jdbc` isn't used at all and can be left out safely when migrating to the new platform.

Beispiel Notebook „Production Coverage“ 2/2

- Abschnitt „Analyse“ führt Berechnungen auf den gegebenen Datenbestand auf und arbeitet die Kernaussage heraus. Eine Visualisierung motiviert die Kernaussage grafisch / managementgerecht.
- Abschnitt „Schlussfolgerung“ fasst das Ergebnis zusammen und führt mögliche nächste Schritte zur Bewältigung des Problems auf.

Nach der Problemlösung kann das Notebook mit neuen Daten noch einmal ausgeführt werden. Dadurch kann geprüft werden, ob die identifizierten Probleme auch behoben wurden.

Choose known tools or tools for plan B.

Welches Tooling ist das richtige?

Die gute Nachricht: Zum Start braucht es überhaupt keine Tools. Erste nachvollziehbare Problemanalysen können auch mit Stift und Papier nach der wissenschaftlichen Methode durchgeführt werden. Die explizite (schriftliche) Darstellung und Diskussion eines Problems hilft meist schon bei der Lösungsfindung.

Erste automatisierte Auswertungen können auch mit bereits bekannten Tools wie Shell-Skripten durchgeführt werden. Die Notebook-Plattform bietet hier entsprechende „Kernels“ zur Code-Ausführung unterschiedlicher Programmiersprachen.

Alternative „Plan B“ ist interessant, wenn man sich sowieso mit den Themen „Big Data“ und „Data Science“ beschäftigen möchte. Hier arbeitet man sich in das Themengebiet allgemein ein und führt dann mit dem neuen Wissen praktische Analysen auf Basis von Softwaredaten durch. Somit wird das neu Gelernte gleich noch einmal vertieft.

STANDARDWERKZEUGE

Jupyter
Interaktives Notizbuch: Zentrale Stelle für Datenanalysen und Dokumentation

Python
Data Scientist's best friend: Einfache, effektiv, schnelle Programmiersprache

Pandas
Pragmatisches Datenanalyse-Framework: Großartige Datenstrukturen und gute Integration mit Machine Learning Tools

D3
JavaScript-Bibliothek für datenorientierte Dokumente: Just beautiful!

Mein Werkzeugkasten

Ich habe mich 2014 dazu entschieden, auf Plan B zu setzen. Daher ist mein Werkzeugkasten sehr Data-Science-lastig.

Als Entwickler verwende ich hier vor allem Python, weil ich schnell und ohne Boilerplate-Code Ergebnisse erzielen kann. Mittlerweile hat sich Python auch als die gesetzte Programmiersprache im Data-Science-Bereich durchgesetzt.

Zusammen mit Jupyter und Pandas sind dadurch schnelle „Wegwerfanalysen“ möglich, aber auch tiefergehende Root-Cause-Analysen. Durch die Mitnutzung des Python-Ökosystems stehen bereits viele Möglichkeiten offen. Zudem kann Python-Code in der Programmiersprache C geschriebene Bibliotheken ansprechen, was sehr effiziente Bibliotheken (z. B. Numpy, TensorFlow, scikit-learn) hervorgebracht hat.

TOOLS for context

Funktionsweise

- Scanne Softwarestrukturen
- Speichere in Graphdatenbank
- Führe Abfragen aus
  - Analysiere Verbindungen
  - Füge Konzepte hinzu
  - Stelle Regeln auf
- Generiere Berichte

jQAssistant

neo4j

GPL Free Software

Free as in Freedom

Der heilige Gral?

Das Werkzeug, welches überhaupt tiefergehende Analysen von Softwaredaten mit all der Vernetzheit erst ermöglicht, ist jQAssistant. jQAssistant ist ein frei verfügbares Framework zur strukturellen Analyse von Softwaredaten. Ursprünglich für Architekturkonfirmitätsanalysen entwickelt, bietet jQAssistant aber auch alles, um verschiedene Datenquellen im Softwarebereich miteinander in Verbindung zu bringen.

jQAssistant scannt Softwareartefakte (Java-Bytecode, Git-Repositories, XML-Dateien etc.) und speichert deren zugrundeliegenden strukturellen Informationen in die Graphdatenbank Neo4j.

Neo4j Schema For Software DATA

Node Labels	Properties	Relationship Types
File	name	CONTAINS
Class	fqn	DEPENDS_ON
Method	signature	INVOKES
Commit	message	CONTAINS_CHANGE

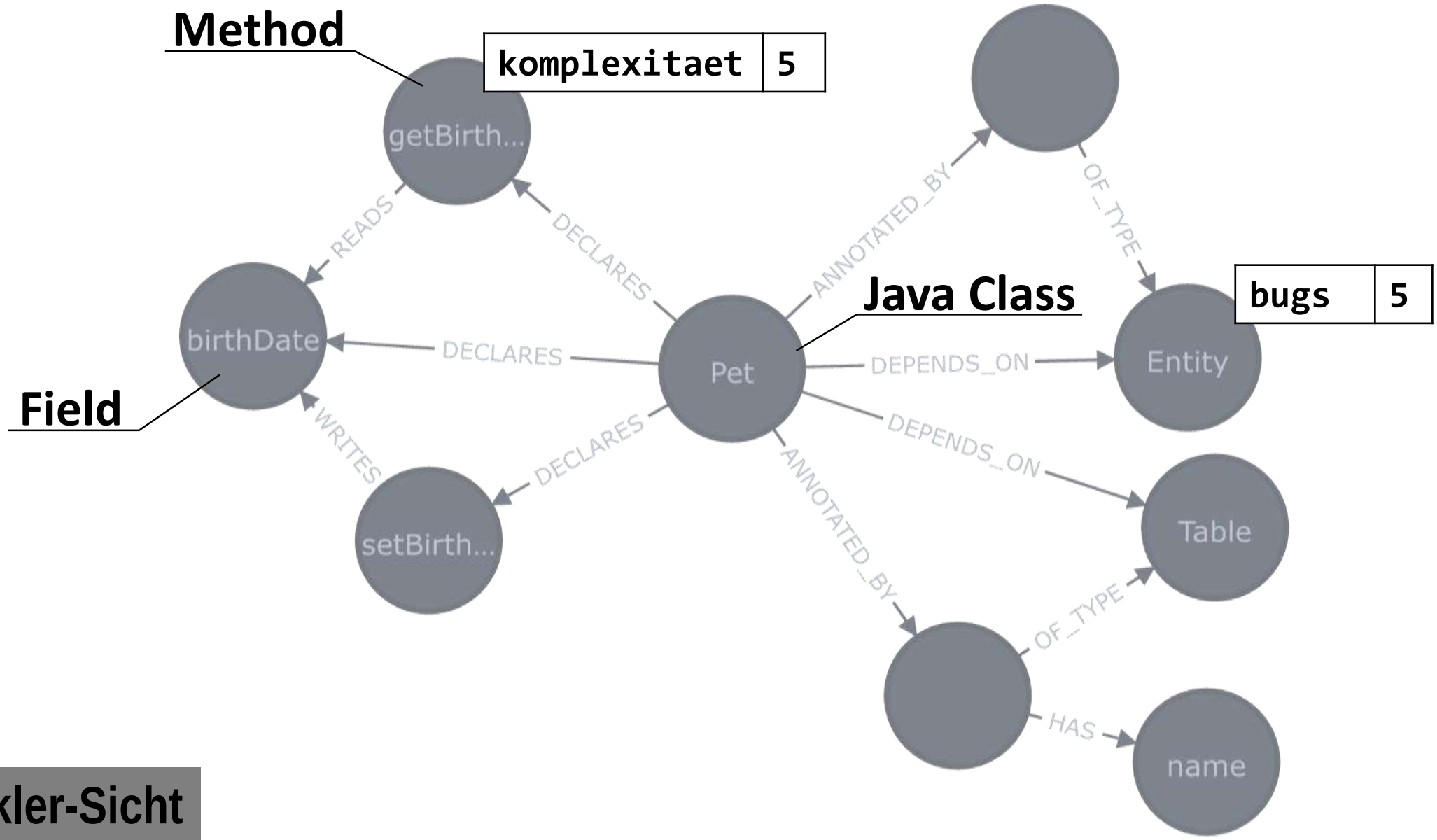
key	value
name	“Pet”
fileName	“Pet.java”
fqn	“foo.bar.Pet”

Schema für Softwaredaten

Werfen wir beispielhalft einen Blick darauf, welche Daten wie in Neo4j abgelegt werden:

- Die einzelnen Informationsfragmente sind als Knoten (nodes) abgelegt und nach ihren Typ mittels Etiketten (labels) markiert
- Zum Knoten zugehörige Daten sind als Eigenschaften (properties) gespeichert
- Zusammenhänge zwischen Knoten werden als Verbindungen (relationships) miteinander verknüpft
- An den Verbindungen selbst können ebenfalls Eigenschaften gespeichert werden

jQAssistant – Die komplexe Softwarelandschaft als Graph



Entwickler-Sicht

Beispiel: Spring PetClinic

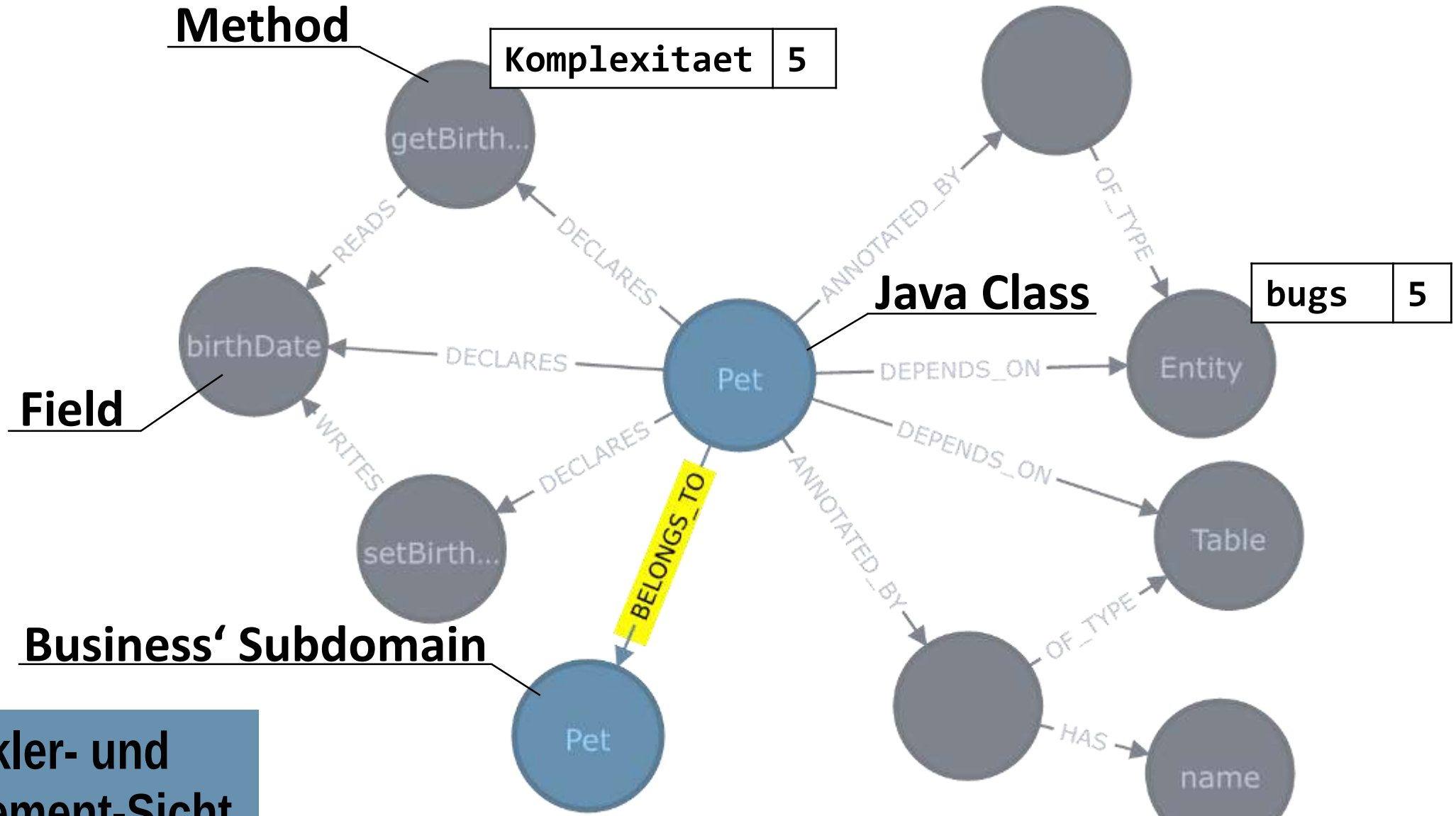
Dieser Teilgraph zeigt das Scan-Ergebnis einer kleinen Java-Webanwendung zur Verwaltung von (Haus-)Tierarztbesuchen.

Wir sehen z. B., dass die Klasse „Pet“ ein Feld „birthdate“ deklariert. Dieses Feld wird von der Methode „getBirthdate“ gelesen. Diese Methode hat eine „Komplexität“ von 5. Die Klasse hat auch eine Beziehung zu einer Annotation des Typs „Entity“, für welche wiederum 5 „Bugs“ identifiziert wurden.

Nun der Clou des Ganzen: Mit der Graphdatenbank-Sprache Cypher können wir nun „höherwertige Informationen“ hinzufügen: Aus der Beziehung zwischen der Klasse „Pet“ und des Typs „Entity“ können wir Schlussfolgern (aka gleich in der Datenbank speichern), dass es sich bei der Klasse „Pet“ um eine Entity gemäß Java Persistence API handelt.

Dadurch haben wir ein neues „Konzept“ einer „JPA-Entity“ abgelegt, welches wir nun für weitere Analysen und Informationsanreicherungen verwenden können. Z. B. könnten wir nun prüfen, ob alle JPA-Entities in einem bestimmten Java-Package abgelegt sind.

jQAssistant – Die komplexe Softwarelandschaft als Graph



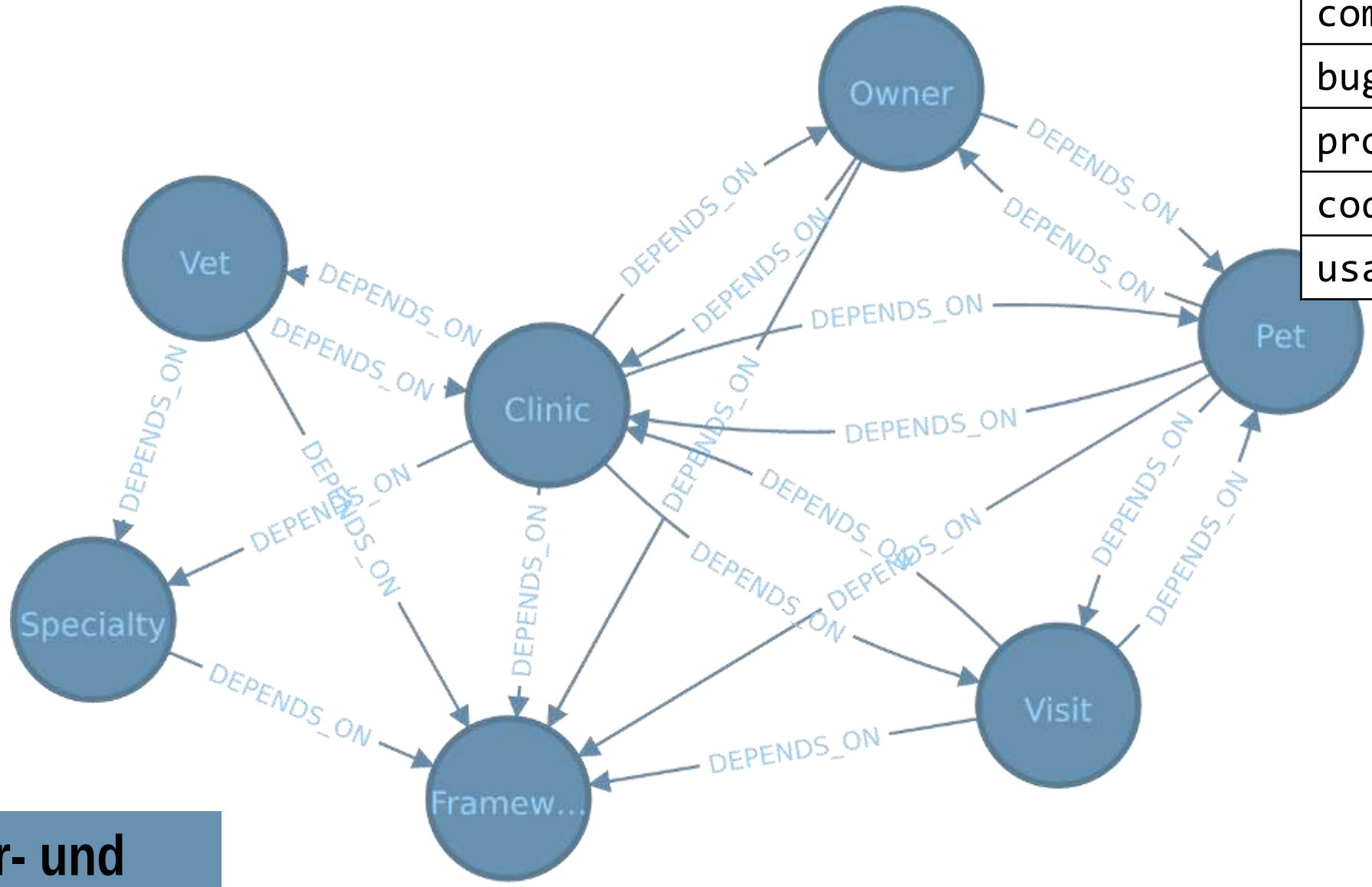
Entwickler- und Management-Sicht

High-Level-Konzepte

Zudem ist es möglich, Defekte in der Anwendung anhand der strukturellen Informationen aufzuspüren. Dies können wir nicht nur für die technischen Aspekte unserer analysierten Software betreiben. Noch spannender wird es, wenn wir uns mit den fachlichen Konzepten einer Anwendung auseinandersetzen. Z. B. könnten wir bestimmte Java-Klassen (z. B. Klasse „Pet“) anhand von Namensschemata einer bestimmten fachlichen Domäne (z. B. Subdomain „Pet“) zuordnen (siehe BELONGS_TO-Beziehung). Hierdurch gelingt uns zum Einen eine Übersetzung von der Technikwelt in die Fachwelt UND zurück.

jQAssistant – Die komplexe Softwarelandschaft als Graph

complexity	1237
bugs	232
problems	25
codechurn	2
usage	86%



Entwickler- und Management-Sicht

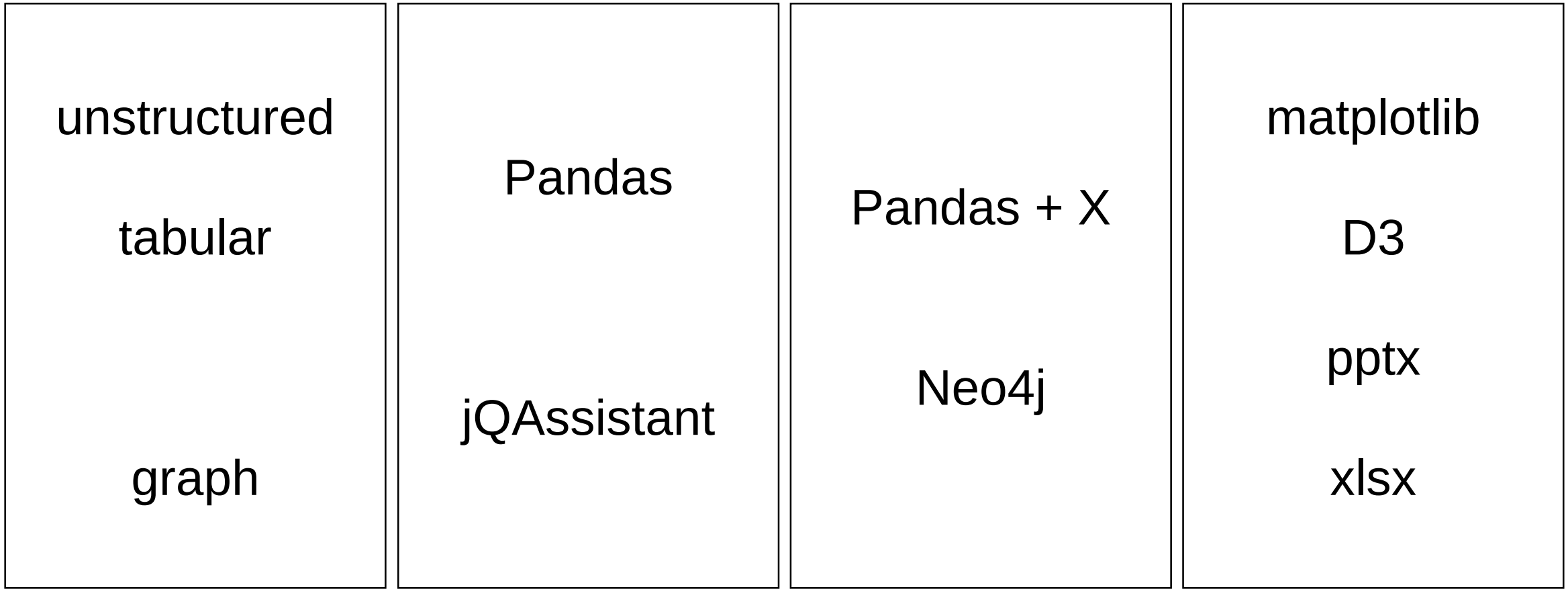
Fachliche Bewertung technischer Defekte

Zum Anderen erreichen wir durch die implizit gegebene Hierarchiebildung, dass wir sehr feingranulare Messungen von Qualitätseigenschaften oder Defiziten auf Code-Ebene auf eine für Entscheidungen angemessene Informationsdichte zusammenfassen können. Information-Overload wird vermieden und technische Qualitätsdefizite werden fachlich diskutierbar.

Zudem stehen einzelnen fachlichen Bereiche durch die zugrundeliegenden Beziehungen der technischen Belangen ebenfalls miteinander in Abhängigkeitsbeziehungen. Dies kann sehr gut dazu dienen, fachlich abgeschlossene Bereiche zu identifizieren (wenn z. B. in „Bounded Context“ gemäß Domain-Driven-Designs identifiziert werden sollen).

THE **JUPYTER** CINEMA

The complete software data analysis pipeline

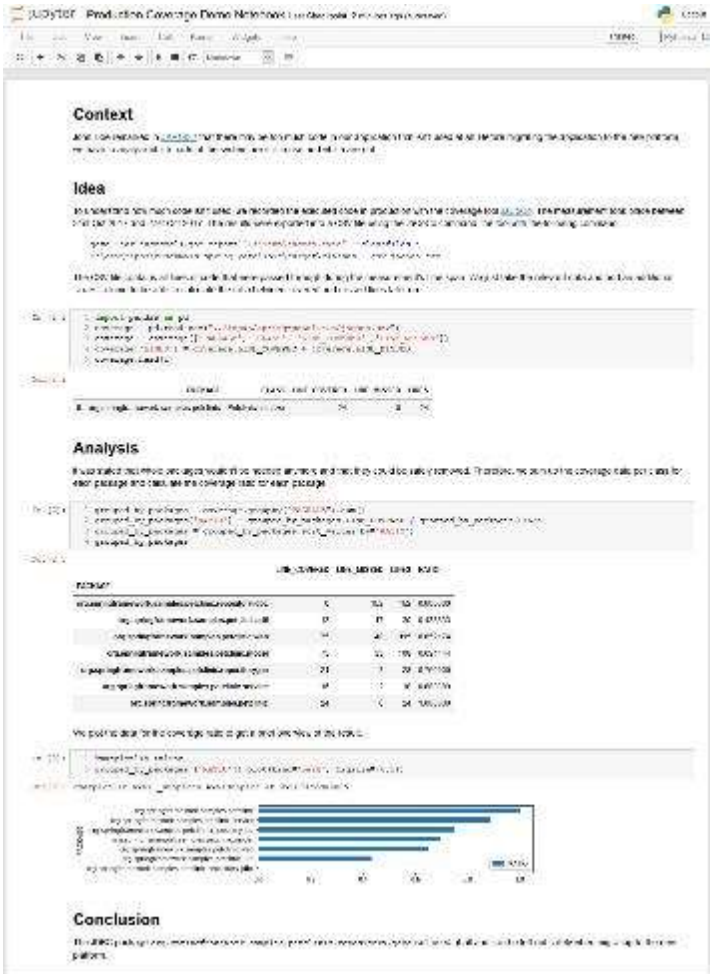


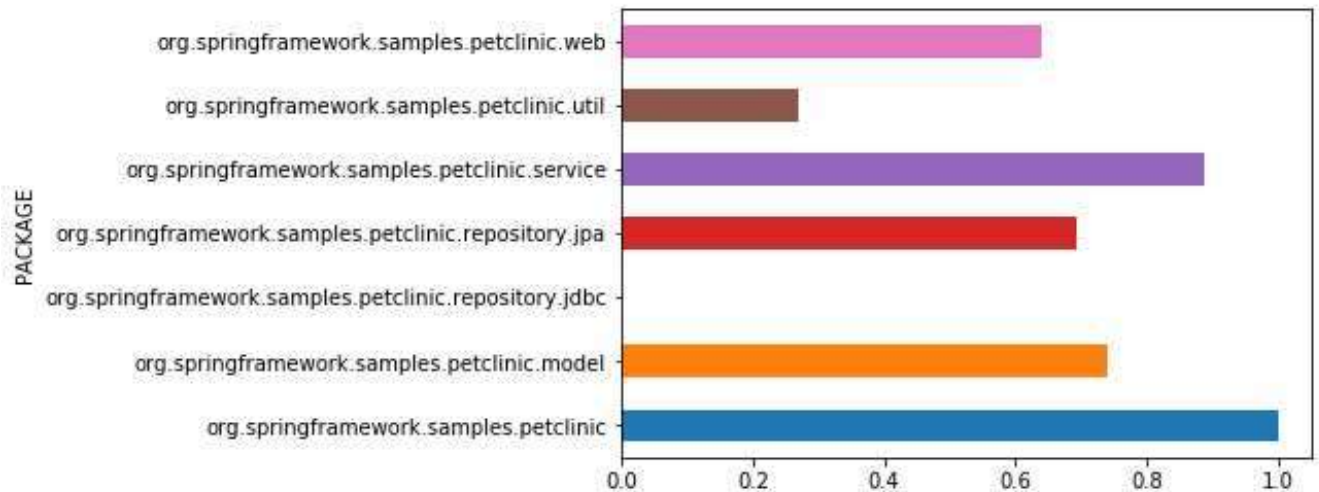
Zusammenhänge innerhalb der Werkzeugkette

- Meine Ausführungsumgebung für Analysen ist das interaktiven Notebook-System Jupyter
- Zum Einlesen textueller, semistrukturierter und tabellarischer Daten verwende ich Pandas. Für graph-artige Daten jQAssistant
- Die Verarbeitung erfolgt je nach Analyse mittels Pandas (+ bei Bedarf weiteren Bibliotheken) und Neo4j im Wechselspiel
- Die Ausgabe erfolgt über eine statische Graphik mittels matplotlib, D3 (bei interaktiven Visualisierungen) oder auch exportiere Listen mit Problemstellen als Excel-Sheets. Zudem können bei Bedarf auch PowerPoint-Folien direkt aus Python generiert werden. Dies ist dann sinnvoll, wenn bestimmte Analysen sich oft wiederholen und „Management-Attention“ genießen sollen

WAS?

Identifikation von wertlosen Codeteilen





https://www.feststelltaste.de/production-coverage-demo-notebook/

Statistiken über die Linux-Kernel-Entwicklung

Hello

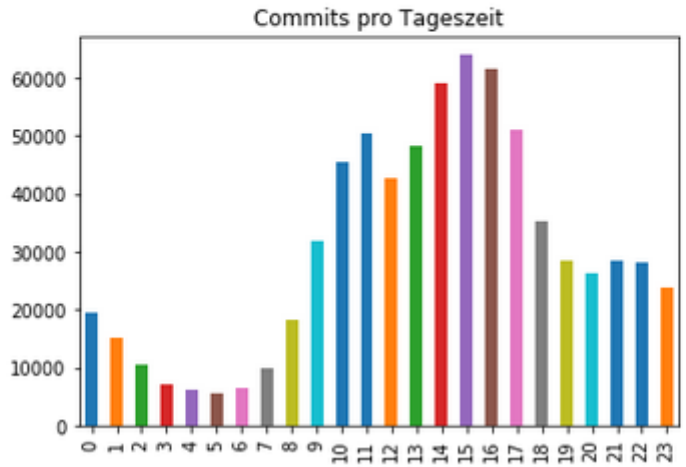
- Analyse

In [6]:
```
1  import pandas as pd
2
3  df = pd.read_csv("../dataset/git_demo_timestamp_linux.csv", sep=";")
4  df.author.value_counts().head(10)
```

Out[6]:
```
Linus Torvalds          24259
David S. Miller          9563
Mark Brown               6917
Takashi Iwai             6293
Al Viro                  6064
H Hartley Sweeten        5942
Ingo Molnar              5462
Mauro Carvalho Chehab    5384
Arnd Bergmann            5305
Greg Kroah-Hartman       4687
Name: author, dtype: int64
```

In [8]:
```
1  df.timestamp_local.dt.hour.value_counts(sort=False).plot(
2      kind='bar',
3      title="Commits pro Tageszeit")
```

Out[8]: `<matplotlib.axes._subplots.AxesSubplot at 0x2893e182c18>`



https://www.feststelltaste.de/developers-habits-linux-edition/

Aggregation von Problemen nach fachlichen Konzepten (Subdomains)

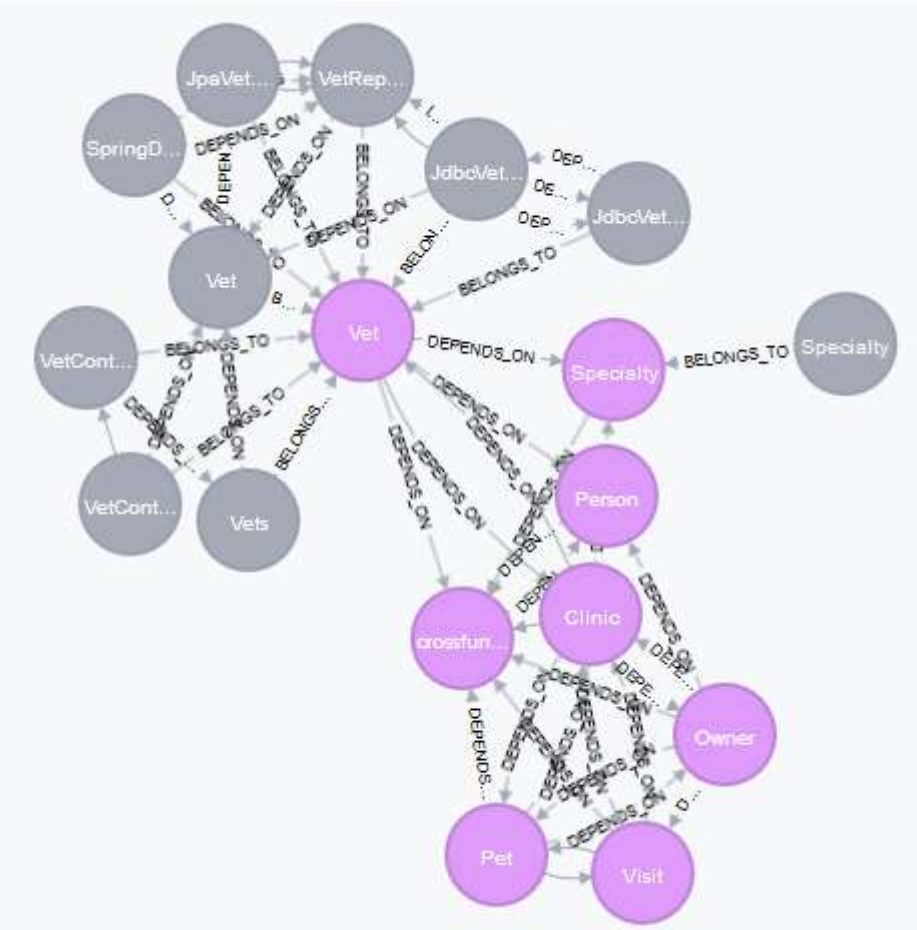
```
// Churn und Bugs per Subdomain
MATCH
  (t:Type)-[:BELONGS_TO]->(s:Subdomain),
  (t)-[:HAS_CHANGE]->(c:Change)
OPTIONAL MATCH
  (t)-[:HAS_BUG]->(b:BugInstance)
RETURN
  s.name as Subdomain,
  COUNT(c) as Changes,
  COUNT(b) as Bugs
ORDER BY Changes DESC
```

$ MATCH (t:Typ...

Rows | Text | Code

Subdomain	Changes	Bugs
Pet	193	22
Owner	157	75
Visit	111	0
Vet	110	0
Clinic	101	0
crossfunctional	99	30
Person	5	0
Specialty	4	0

Started streaming 8 records after 415 ms and comple



Artikel noch nicht online, Grundlagen jedoch hier:
https://www.feststelltaste.de/building-higher-level-abstractions-of-source-code/

Zusammenfassung

Bekämpfe Risiken!

+ mache sie sichtbar und verständlich
+ erzeuge sachliche Diskussionen

Treibe Entscheidungen!

10 Priorisiere wertvolle Verbesserungen
20 Zeige, dass es besser wird; GOTO 10

Take Action!

+ Meistere Herausforderungen gemeinsam
+ Fange einfach einfach an :-)

Zum Mitnehmen

Es ist absolut notwendig, Risiken in der eigenen Software klar und deutlich zu kommunizieren. Wir Softwareentwickler sind besonders von den zwei fundamentalen Problemen betroffen, die beim Umgang mit Wissen und Entscheidungen auftreten. Software Analytics kann hier helfen, sachliche Diskussionen über nun nicht mehr als richtig anzusehenden Entscheidungen aus der Vergangenheit entstehen zu lassen. Wichtig ist, darüber reden zu können!

Durch erfolgreiche Analysen (und der nachfolgenden Problemlösung) wächst die Erfahrung und Wahrscheinlichkeit, Budget für neue Analysen bewilligt zu bekommen. Dies ist genau das Gegenteil der ansonsten üblichen „Todesspiralen“ in der Softwareentwicklung.

Eigene, erste Schritte sind einfach möglich. Das passende Tooling kann sich nach und nach selbst angeeignet werden. Wichtig ist der offene Ansatz bei den Datenanalysen. Dadurch ist es möglich, miteinander aus den vorgenommenen Analysen zu lernen.

Fange also einfach einfach an!

Literaturempfehlung

akademisch

Christian Bird, Tim Menzies, Thomas Zimmermann:
The Art and Science of Analyzing Software Data

Tim Menzies, Laurie Williams, Thomas Zimmermann:
Perspectives on Data Science for Software Engineering

praktisch

Wes McKinney: Python For Data Analysis

Adam Tornhill: Software X-Ray

Buchempfehlungen

- Wer sich gerne mit den akademischen Ursprüngen von Software Analytics beschäftigen möchte, findet bei den beiden ersten Büchern eine wahnsinnige Dichte an Informationen.
- Das Buch von Wes McKinney stellt allgemein das Datenanalyse-Framework Pandas vor.
- Das Buch von Adam Tornhill ist meines Erachtens eines der besten und praktischsten Bücher für den Einstieg in die Analyse von Daten im Softwarebereich. Es ist zwar sehr produktzentriert, aber die zugrundeliegenden Praktiken werden offengelegt. Ich selbst war bei dem Buch einer der technischen Reviewer und habe hier ordentlich Feedback gegeben, dass auch in das Buch mit eingeflossen ist.