SOFTWARE ARCHITEKTUREN GEZIELT ANALYSIEREN, DOKUMENTIEREN UND VERBESSERN MIT SOFTWARE ANALYTICS

TU Dresden, Vorlesung „Softwareentwicklung in der industriellen Praxis“

20. Januar 2020, Dresden

Stephan Pirnbaum @spirnbaum

BUSCHMAIS

BUSCHMAIS ist ein Dresdner IT-Beratungsunternehmen, gegründet im Jahre 2008. Unsere Schwerpunkte liegen in der Architekturberatung und der Entwicklung moderner Geschäftsanwendungen.

Gemeinsam mit unseren Kunden analysieren, planen und optimieren wir IT-gestützte Prozesse und unterstützen sie bei der Umsetzung neuer Anforderungen. Dabei arbeiten wir branchenunabhänig und technologiefokussiert in effizienten Teams.

BUSCHMAIS GbR

Leipziger Straße 93

01127 Dresden

Tel. +49 351 3209230

info@buschmais.com

www.buschmais.de

Wenn ich diesen Code ändere, weiß ich nicht, was ich sonst noch einreiße!

Eine Neuimplementierung ist schneller als eine Änderung des bestehenden Codes!

Die Komplexität der Umsetzung eines Features ist höher als die fachliche Komplexität!

Test- und Releaseprozesse sind langwierig und fehleranfällig!

WARUM?

Die Anwendung lässt sich nicht horizontal skalieren!

Das Anlernen neuer Kollegen bindet zu viele Ressourcen!

Die Entwicklung verschiedener Domänen lässt sich nicht auf unterschiedliche Teams verteilen!

UM DIE PROBLEME ZU LÖSEN,
MÜSSEN WIR DIE URSACHEN VERSTEHEN!

ES WAR EINMAL...

ein Greenfield Projekt

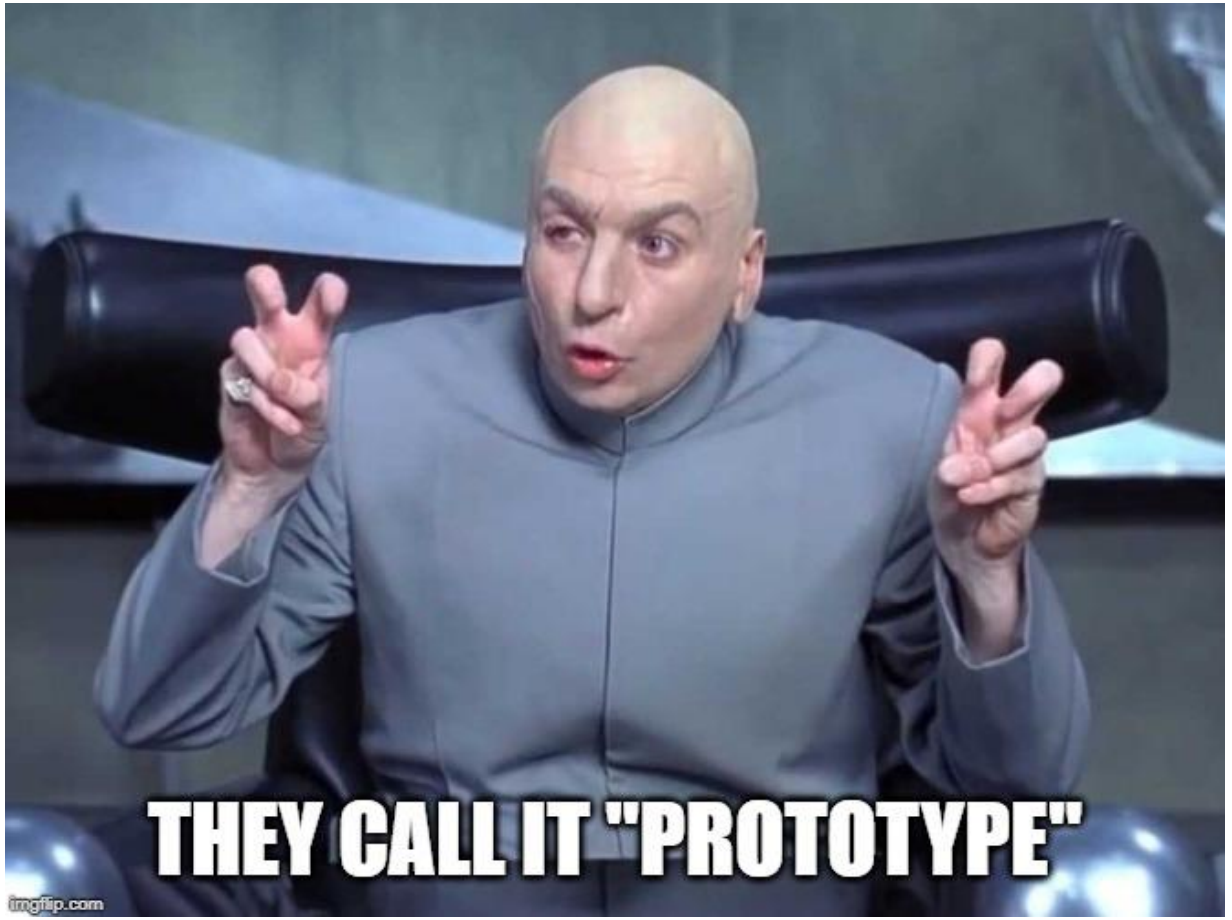
- geringe fachliche Komplexität
- kleine Source Code Basis
- leichte Anpassbarkeit des Codes
- Chancen

ES WAR EINMAL...

ein Greenfield Projekt

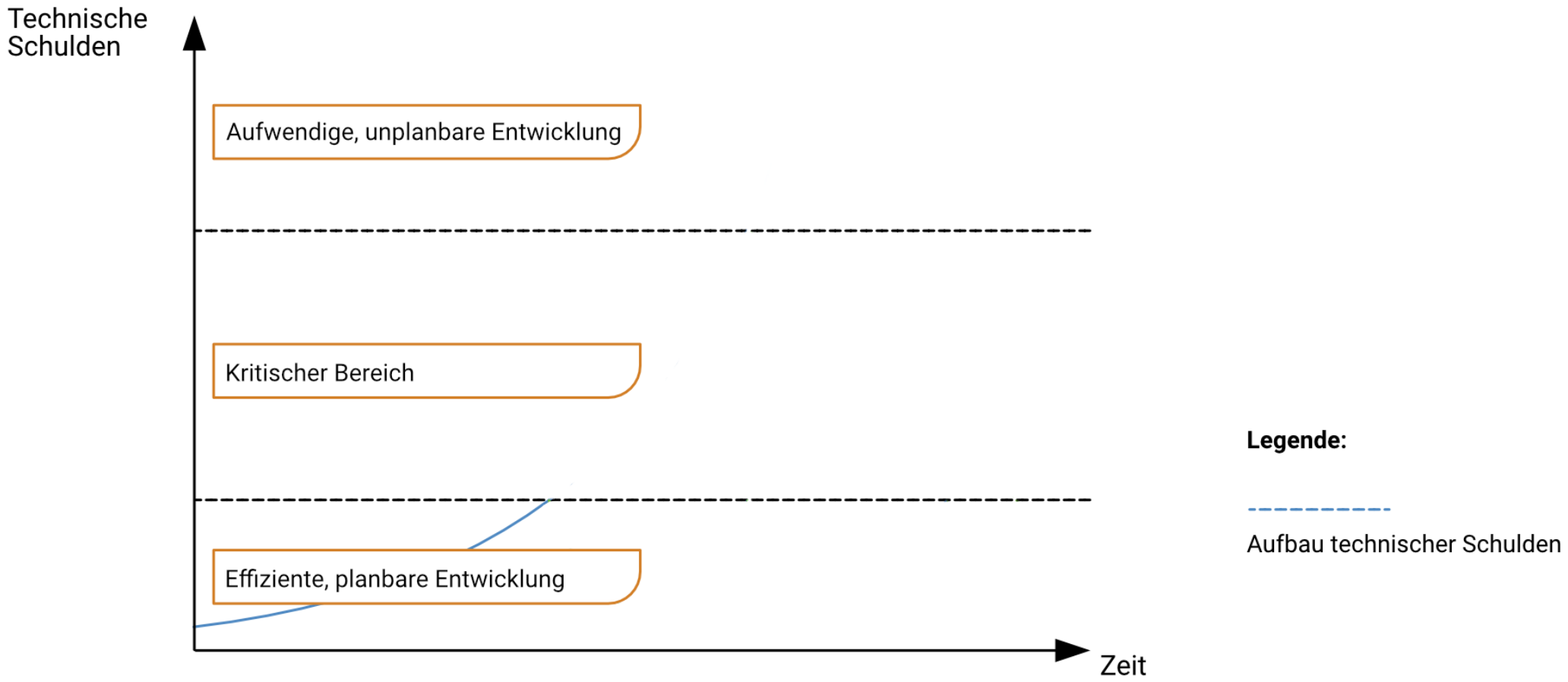
- geringe Erfahrung mit den Anwendungsfällen
- unbekannte und unklare Anforderungen
- implizite und explizite Architekturdefinition notwendig
- Risiken

ES WAR EINMAL…



https://imgflip.com/i/3hp50v

ES WAR EINMAL...



DIE GESCHICHTE SCHREIBT SICH FORT…

http://www.quickmeme.com/p/3vpkh8

DIE GESCHICHTE SCHREIBT SICH FORT…

Vom Greenfield zum Brownfield

- Fließender Übergang von Prototyp zu Produktivsystem
- Funktionelles Wachstum der Anwendung
  - Steigende fachliche Komplexität
  - Sichtbarwerden von technischen/architektonischen Limitierungen

DIE GESCHICHTE SCHREIBT SICH FORT…

Vom Greenfield zum Brownfield

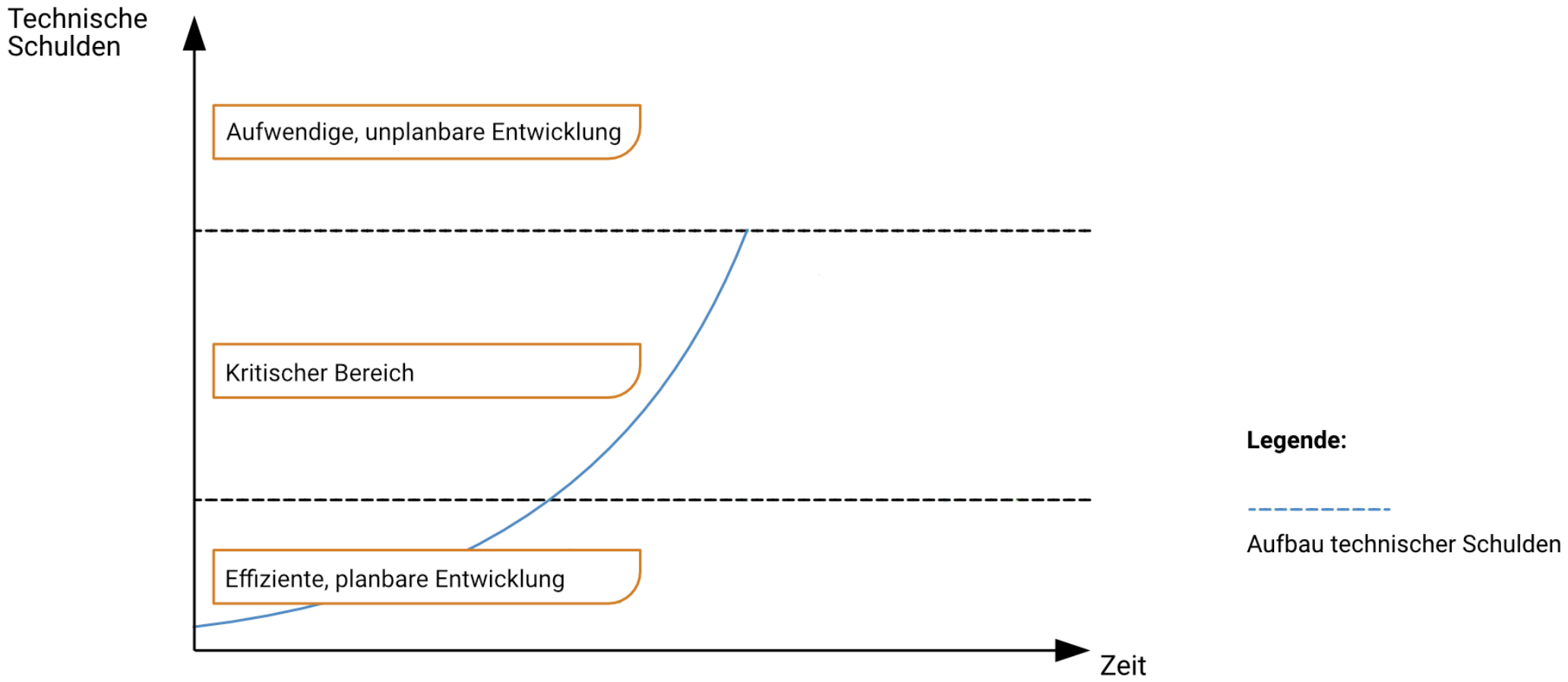
- Organisatorisches Wachstum und Fluktuation
  - Entstehen von Knowledge Loss und Blindspots
- Technisches Wachstum der Anwendung
  - Steigende Komplexität durch neue Frameworks und Technologien

DIE GESCHICHTE SCHREIBT SICH FORT…

Vom Greenfield zum Brownfield

- Wachstum der Anwendung ist entscheidend
  - Schnelles, Quantitatives Wachstum → Stärkere Architekturerosion
  - Langsames, Qualitatives Wachstum → Geringere Architekturerosion

DIE GESCHICHTE SCHREIBT SICH FORT…

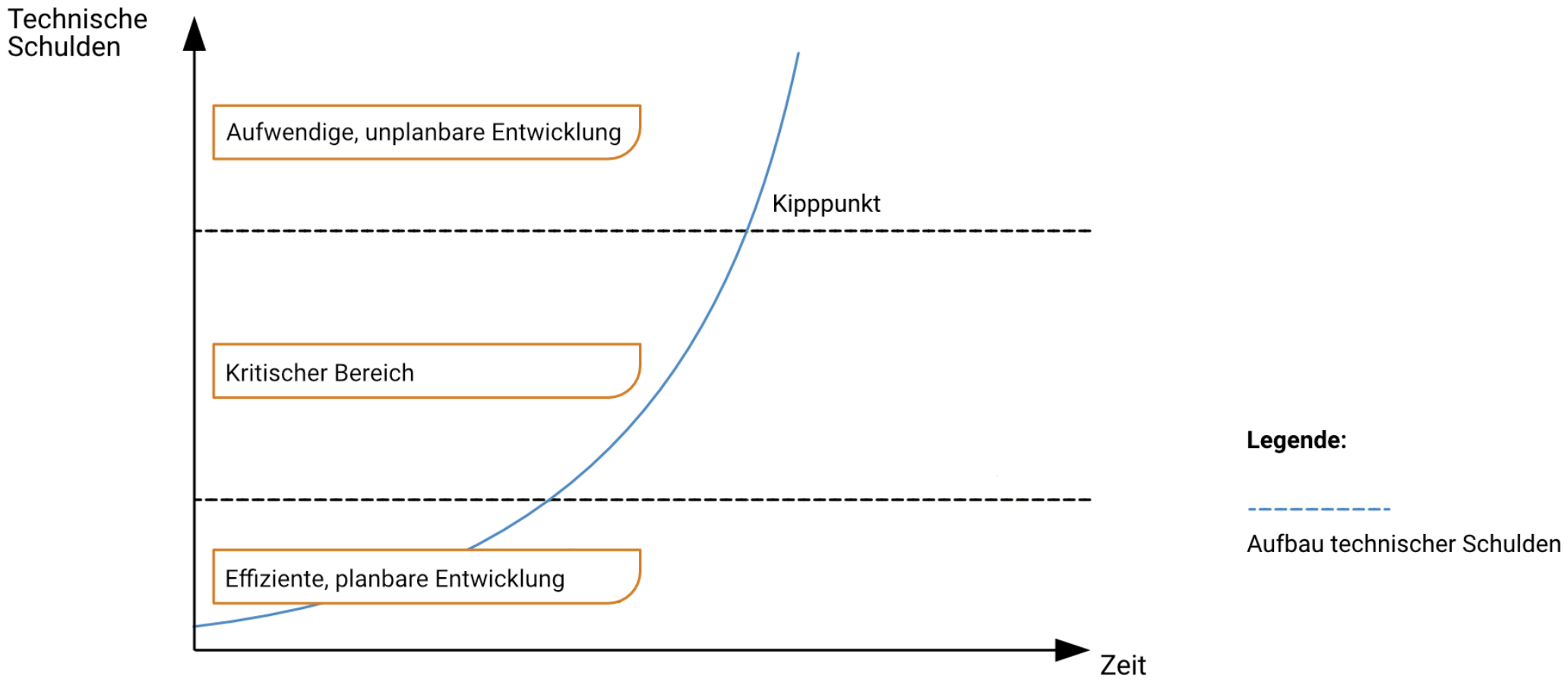


DIE GESCHICHTE SCHREIBT SICH FORT…

- Innovationszwang durch externe Faktoren
  - Neue Konkurrenten
  - Steigende Kundenanforderungen
  - Andere Geschäftsmodelle

Kann unser System folgen?

DIE GESCHICHTE SCHREIBT SICH FORT…



DIE GESCHICHTE SCHREIBT SICH FORT…

Probleme

- Featurekomplexität vs. Umsetzungskomplexität

- Erschwerte Projektplanung
- Lange Release-Zyklen

DIE GESCHICHTE SCHREIBT SICH FORT…

Probleme

- Inselwissen und unzureichende Dokumentation
- Komplexe Code-Strukturen

- Wissenstransfer schwierig
- Fluktuation/Krankheit von Mitarbeitern riskant

DIE GESCHICHTE SCHREIBT SICH FORT…

Probleme

- Technische Limitierungen durch eingesetzte Frameworks
- Komplexer Build-, Test-, Releaseprozess

- Mangelnde Skalierbarkeit

DIE GESCHICHTE SCHREIBT SICH FORT…

Probleme

Dokumentierte Architektur (Wiki, PowerPoint, …)

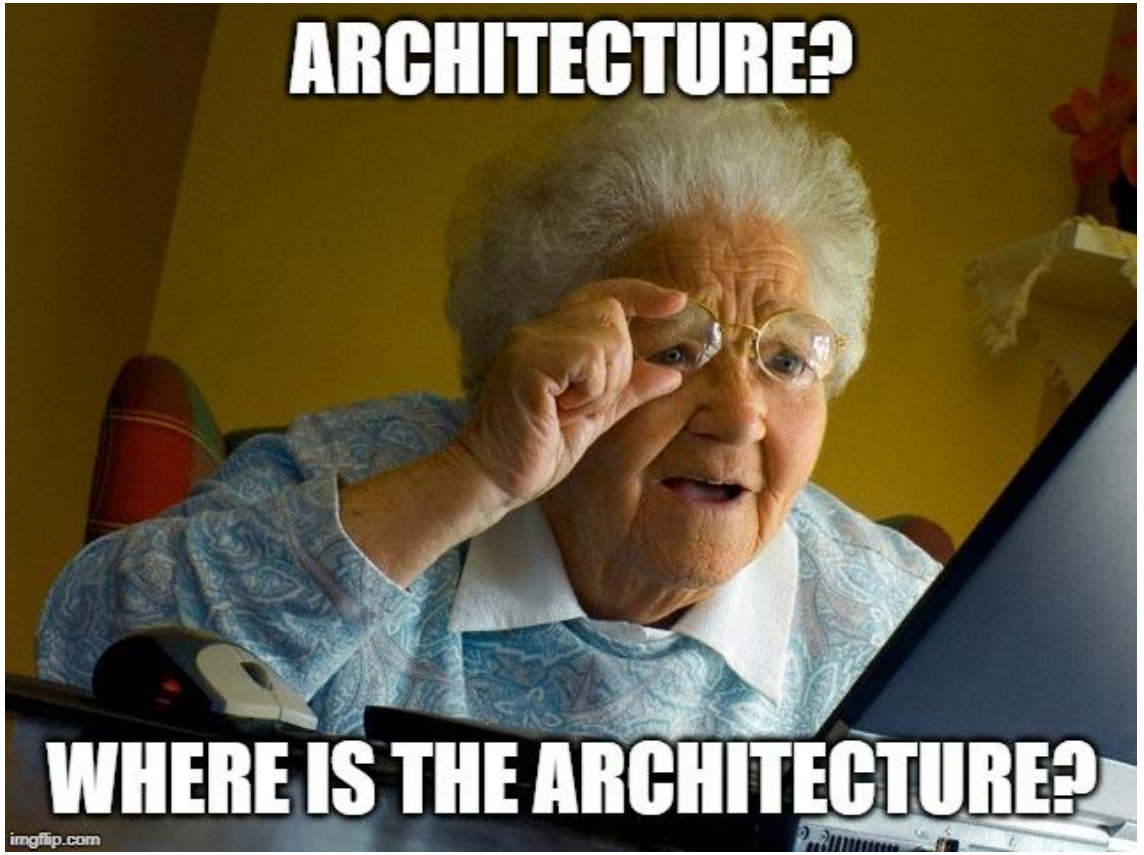
<>

Gefühlte Architektur (Entwickler)

<>

Reale Architektur (Code)

DIE GESCHICHTE SCHREIBT SICH FORT…



https://imgflip.com/i/3db8rk

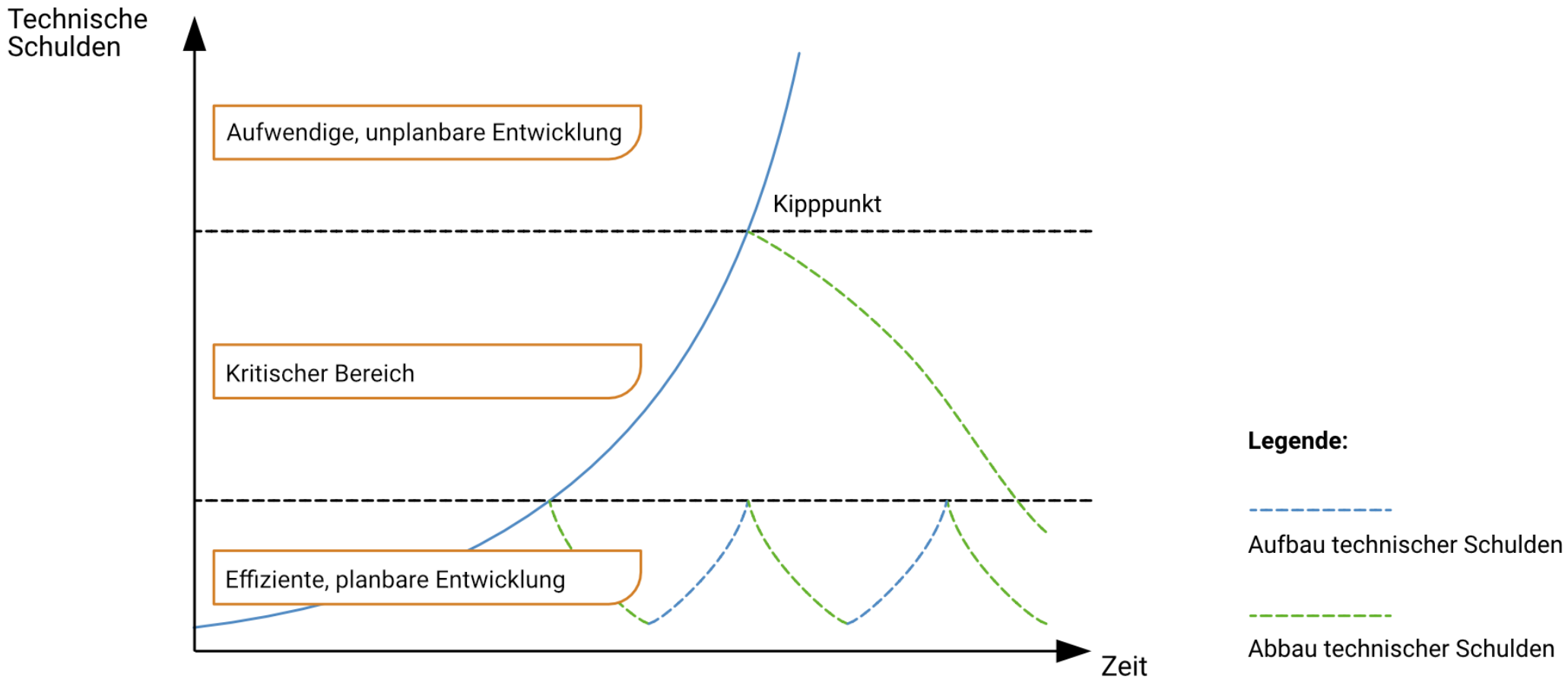
ABER...

Legacy Systeme: Eine Medaille – Zwei Seiten

- Bewährtes Rückgrat betrieblicher Abläufe
- Hoher Business Value

- Hohe Komplexität
- Hoher Entwicklungsaufwand

WIE KOMME ICH DA WIEDER RAUS?



WIE KOMME ICH DA WIEDER RAUS?

- aktives Management von technischen Schulden
  - unter Berücksichtigung der kurz- und langfristigen Ziele
  - als fester Bestandteil der Projektplanung
  - gezielt bei kritischen Bereichen

WIE KOMME ICH DA WIEDER RAUS?

Neu-Implementierung

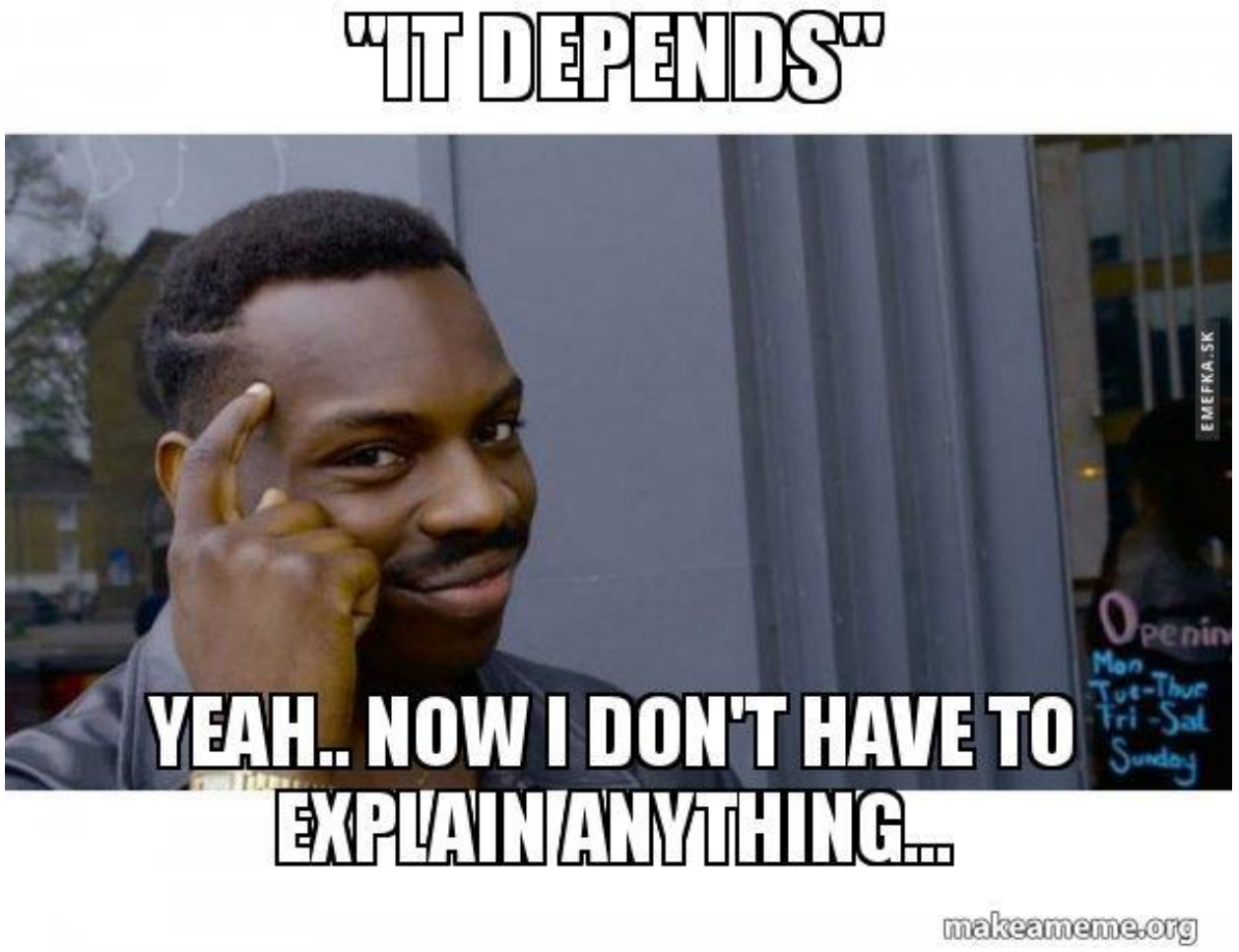
vs.

Zerlegung in Microservices

vs.

Restrukturierung des Monolithen

WIE KOMME ICH DA WIEDER RAUS?



WIE KOMME ICH DA WIEDER RAUS?

WICHTIG

Proper Preparation Prevents Poor Performance

WIE KOMME ICH DA WIEDER RAUS?

EIN BEISPIEL

- Open-Source e-Commerce System „shopizer“
  - Warenkorb
  - Katalog
  - Suche
  - Bestellung
  - …
- Fork: https://github.com/buschmais/shopizer

EIN BEISPIEL

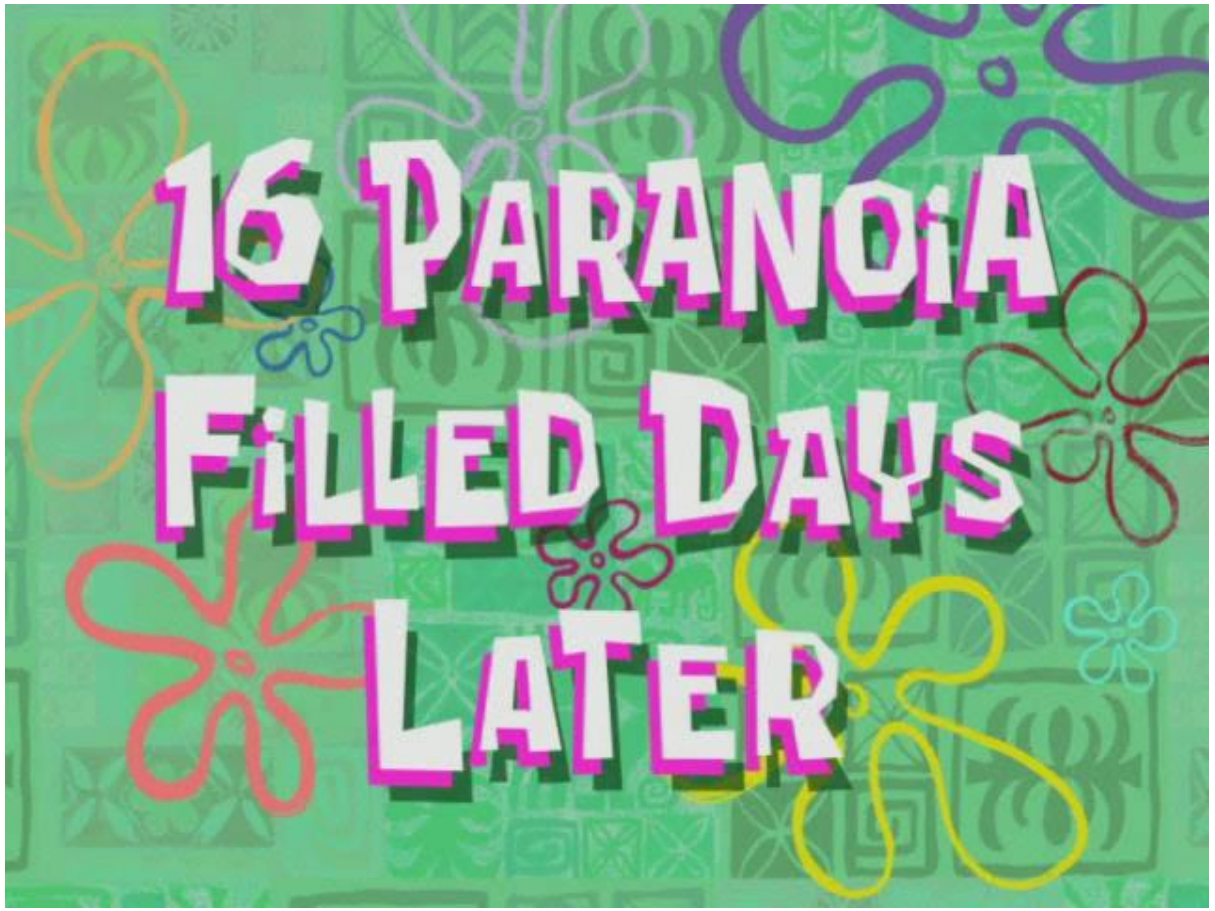
- Szenario (Organisatorisch)
  - Shop soll stark anwachsen (Funktionsumfang, Nutzerbasis)
    - Verkauf digitaler Produkte
  - Onboarding zahlreicher neuer Entwickler
    - Strukturierung in fachliche Teams
  - Vorbereitung soll eingeplant und umgesetzt werden

EIN BEISPIEL

- Ziel
  - Identifikation von Wissensträgern
  - Abbau und aktives Management technischer Schulden
  - Refaktorisierung der Strukturen hin zu fachlichen Schnitten
  - Erstellung von Dokumentation

EIN BEISPIEL

- Problem
  - Verteiltes und fehlendes Wissen über Anwendungsstruktur
  - Architektur- und Designentscheidungen nicht nachvollziehbar
  - Unbekanntes Nichtwissen
  - Zeitrestriktionen
  - …



https://spongebob.fandom.com/wiki/List_of_time_cards

https://imgflip.com/memegenerator/Skeptical-Baby

https://builder.cheezburger.com/Builder/RenderPreview/65f84df1-bae0-4cb8-9602-5205a7caf98d

SOFTWARE ANALYTICS

SOFTWARE ANALYTICS

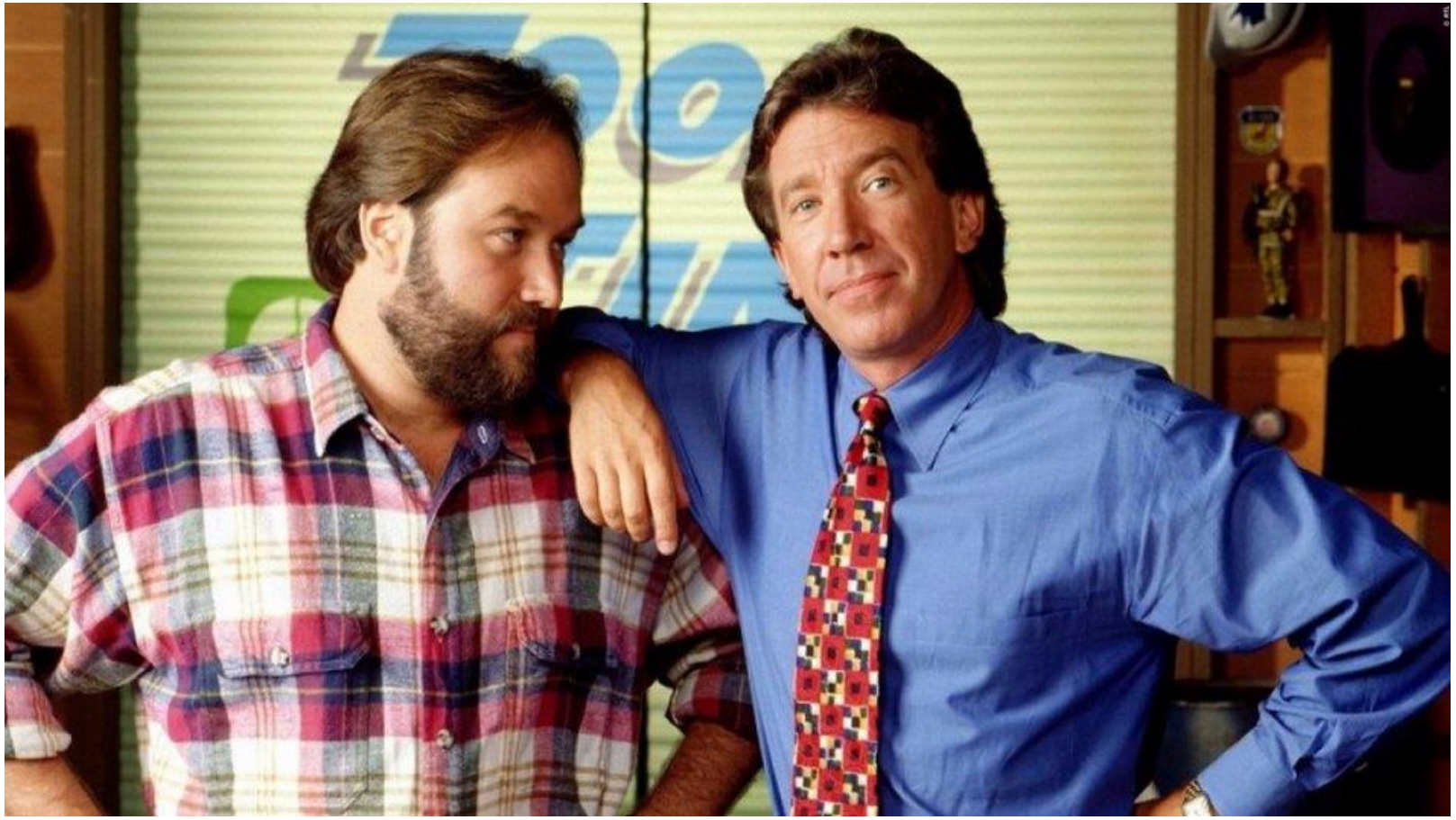
SOFTWARE ANALYTICS

- Data Analytics für Software Systeme
  - Extraktion und Zusammenführung von Informationen aus
    - Source Code
    - Statischen und dynamischen Eigenschaften
    - Entwicklungshistorien
  - Schlussfolgern von neuen Informationen zum Aufbau von Wissen

SOFTWARE ANALYTICS

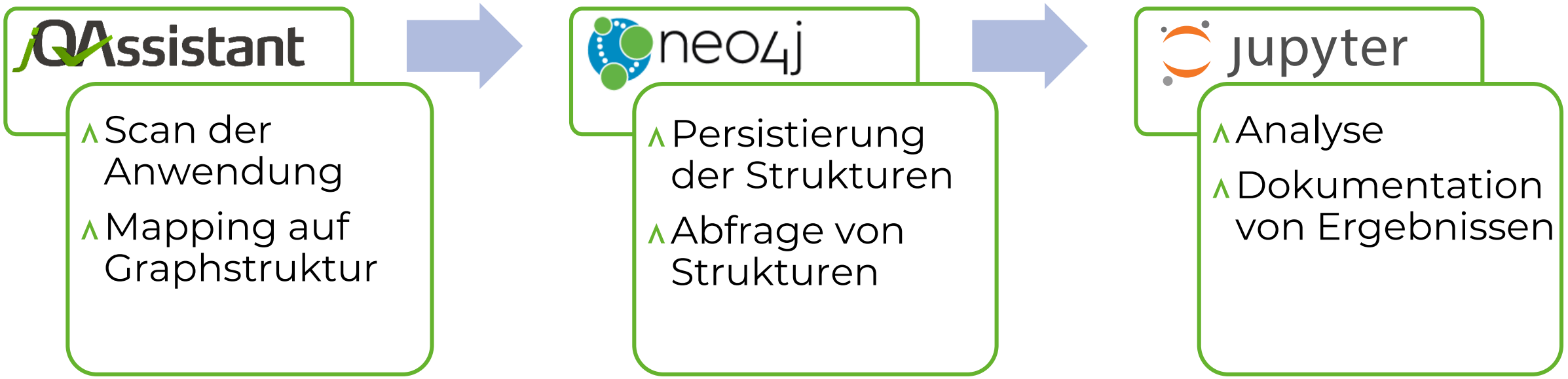
- Dokumentation von Erkenntnissen
  - Unterstützung von Planung und Refaktorisierung
  - Referenz für (neue) Entwickler
  - Kommunikationsgrundlage für Entwickler und Entscheider

SOFTWARE ANALYTICS

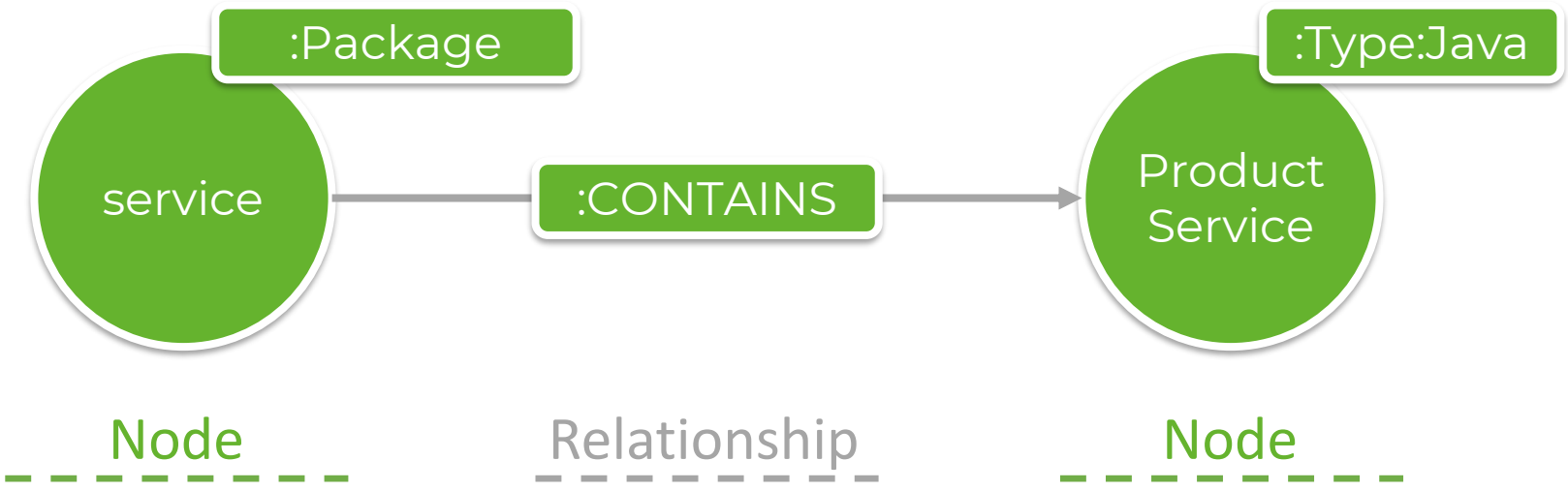


https://www.kino.de/serie/hoer-mal-wer-da-haemmert-1999/news/hoer-mal-wer-da-haemmert-tim-allen-bringt-neue-folgen-ins-gespraech/

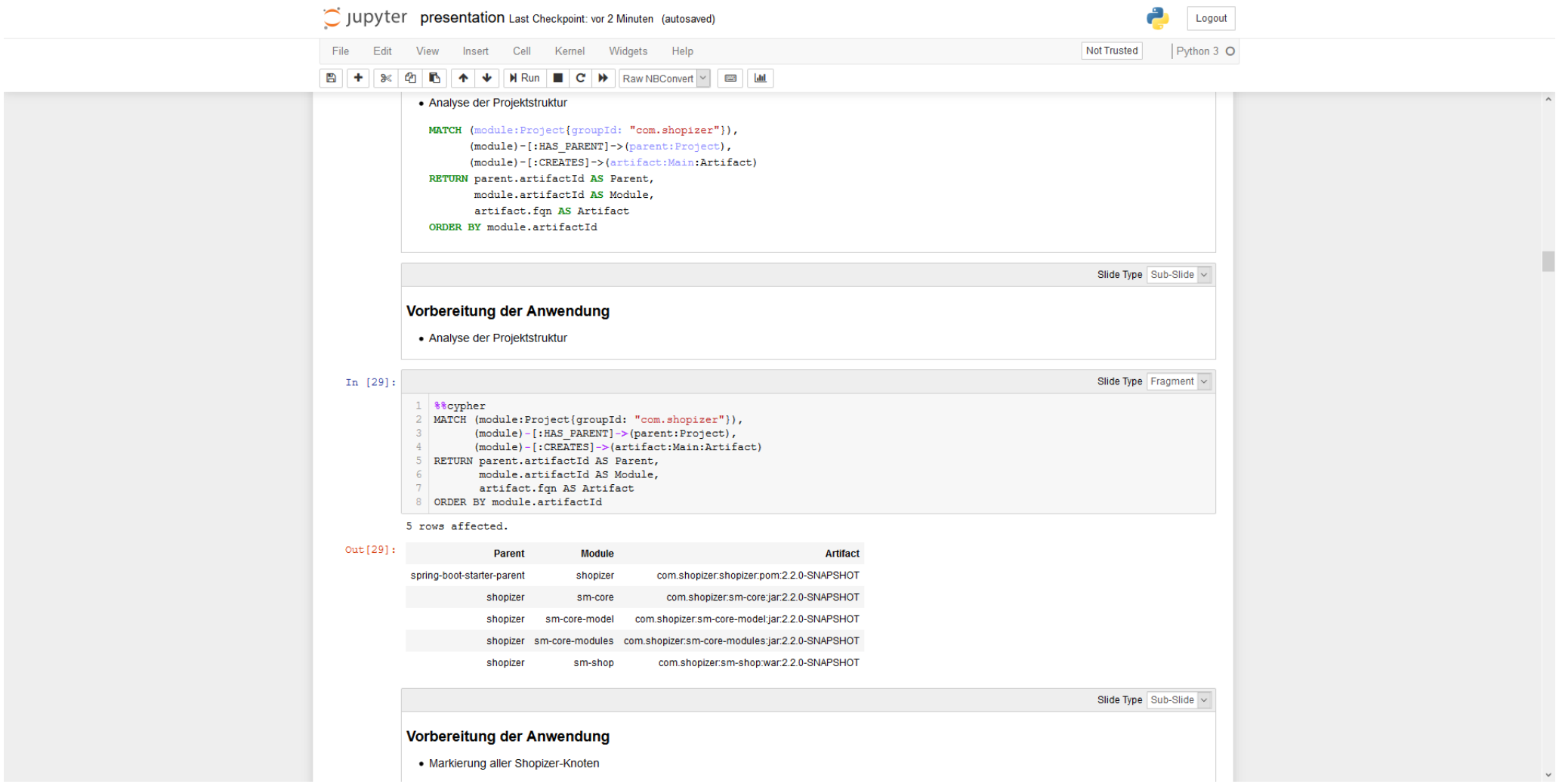
SOFTWARE ANALYTICS



… MIT JQASSISTANT UND NEO4J



… MIT JUPYTER


jupyter presentation Last Checkpoint: vor 2 Minuten (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Not Trusted
Python 3
Run
Raw NBConvert
• Analyse der Projektstruktur
MATCH (module:Project{groupId: "com.shopizer"}),
(module)-[:HAS_PARENT]->(parent:Project),
(module)-[:CREATES]->(artifact:Main:Artifact)
RETURN parent.artifactId AS Parent,
module.artifactId AS Module,
artifact.fqn AS Artifact
ORDER BY module.artifactId
Slide Type Sub-Slide
Vorbereitung der Anwendung
• Analyse der Projektstruktur
In [29]:
Slide Type Fragment
%%cypher
MATCH (module:Project{groupId: "com.shopizer"}),
(module)-[:HAS_PARENT]->(parent:Project),
(module)-[:CREATES]->(artifact:Main:Artifact)
RETURN parent.artifactId AS Parent,
module.artifactId AS Module,
artifact.fqn AS Artifact
ORDER BY module.artifactId
5 rows affected.
Out[29]:
Parent Module Artifact
spring-boot-starter-parent shopizer com.shopizer:shopizer:pom:2.2.0-SNAPSHOT
shopizer sm-core com.shopizer:sm-core:jar:2.2.0-SNAPSHOT
shopizer sm-core-model com.shopizer:sm-core-model:jar:2.2.0-SNAPSHOT
shopizer sm-core-modules com.shopizer:sm-core-modules:jar:2.2.0-SNAPSHOT
shopizer sm-shop com.shopizer:sm-shop:war:2.2.0-SNAPSHOT
Slide Type Sub-Slide
Vorbereitung der Anwendung
• Markierung aller Shopizer-Knoten

SOFTWARE ANALYTICS

https://makeameme.org/meme/possibilities-are-endless-59fa8f

SOFTWARE ANALYTICS

Wie ist die Anwendung technisch strukturiert?

Wie ist die Anwendung fachlich strukturiert?

Was sind änderungskritische Anwendungsteile?

SOFTWARE ANALYTICS

Vorbereitung der Anwendung

SOFTWARE ANALYTICS

```
<plugin>
  <groupId>com.buschmais.jqassistant</groupId>
  <artifactId>jqassistant-maven-plugin</artifactId>
  <version>1.7.0</version>
  <executions><execution>
      <goals>
        <goal>scan</goal>
        <goal>analyze</goal>
      </goals>
  </execution></executions>
  <.../>
</plugin>
```

SOFTWARE ANALYTICS

```
mvn clean install jqassistant:server
```

→ `localhost:7474`

SOFTWARE ANALYTICS

https://www.memecreator.org/meme/lets-get-to-work81

SOFTWARE ANALYTICS

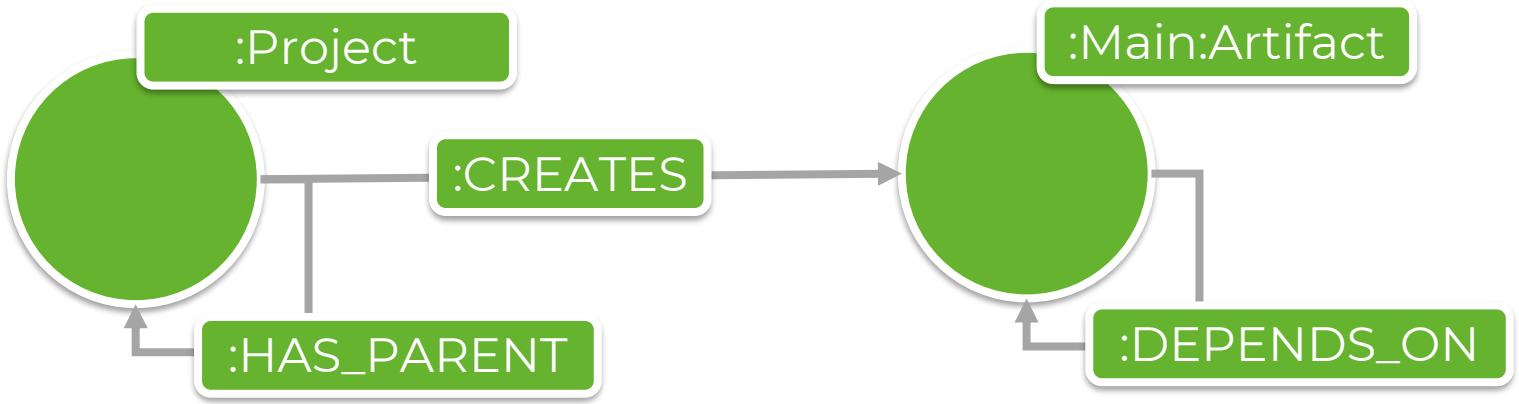
Wie ist die Anwendung technisch strukturiert?

SOFTWARE ANALYTICS

- Analyse der technischen Strukturierung
  - Umsetzung bekannter Architekturprinzipien- und Muster
  - Abgleich von dokumentierter und realer Architektur
    - Aktualisierung der Dokumentation zur Wissensverteilung
  - Planung gezielter Schritte zum Abbau technischer Schulden

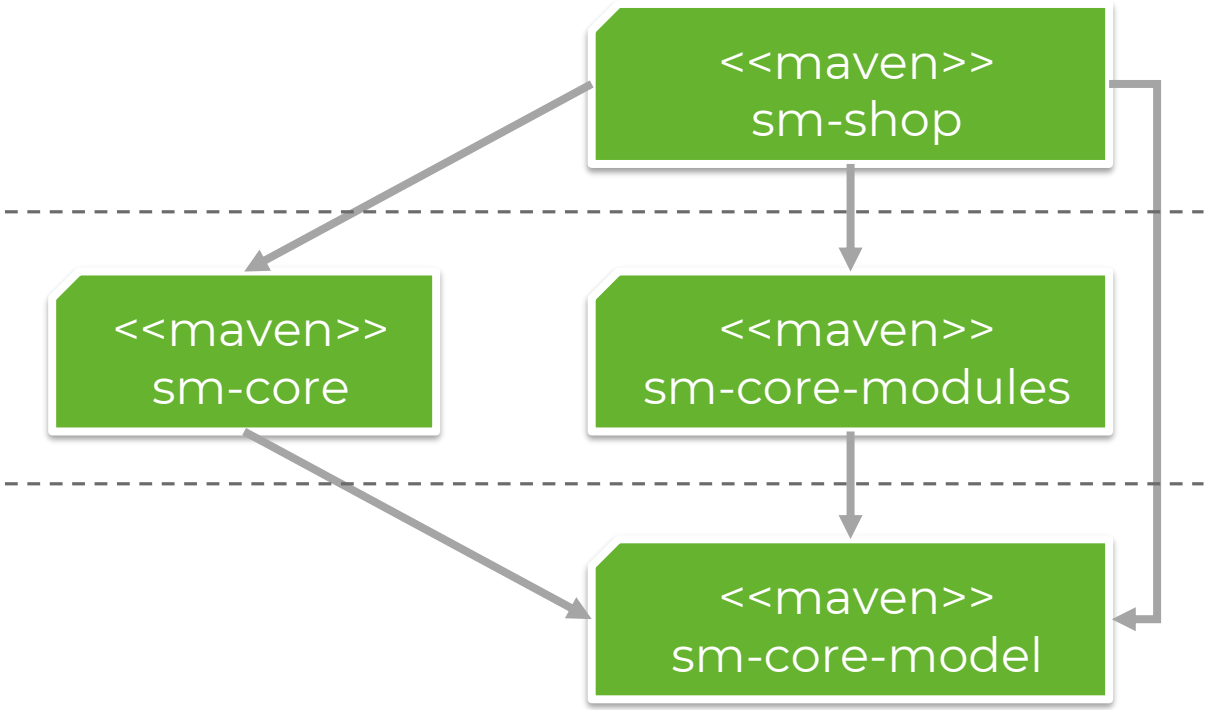
SOFTWARE ANALYTICS

- Analyse der technischen Strukturierung



SOFTWARE ANALYTICS

- Live-Demo
  - Abhängigkeit zwischen maven-Modulen
  - Zuordnung von Artefakten zu Layern (Presentation, Domain, Data Layer)



SOFTWARE ANALYTICS

Wie ist die Anwendung fachlich strukturiert?

SOFTWARE ANALYTICS

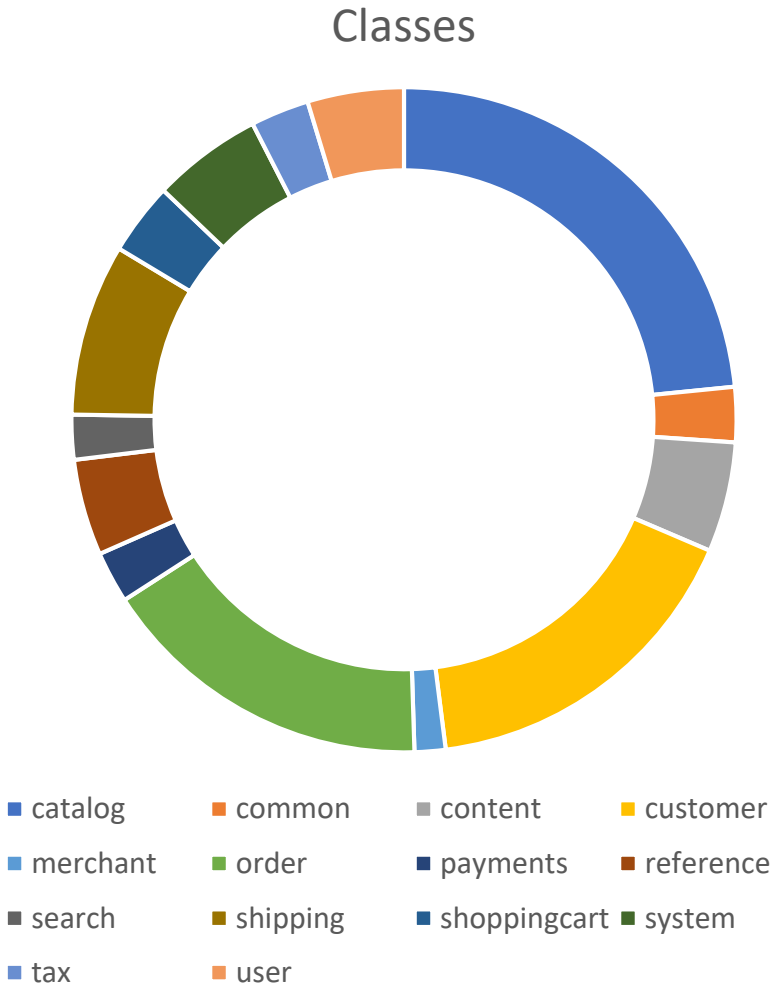
- Analyse der fachlichen Strukturierung
  - Definition und Dokumentation einer fachlichen Strukturierung
    z.B. nach DDD
  - Separierung von Source Code nach fachlicher Zugehörigkeit
  - Fokussierung der Entwickler auf kleinere, kohäsive Anwendungsteile
  - Planung und Umsetzung konkreter Refaktorisierungsschritte

SOFTWARE ANALYTICS

- Wo ist eine gute fachliche Strukturierung am ehesten zu erwarten?
  - Spring-Services
  - Paketnamen (`service`, konkrete Fachlichkeiten wenn bekannt)
  - Explorieren der Anwendung in IDE

SOFTWARE ANALYTICS

- Live-Demo
  - Fachliche Packages (Subdomänen)
  - Zuordnung von Klassen zu Subdomänen anhand Package-Namen



SOFTWARE ANALYTICS

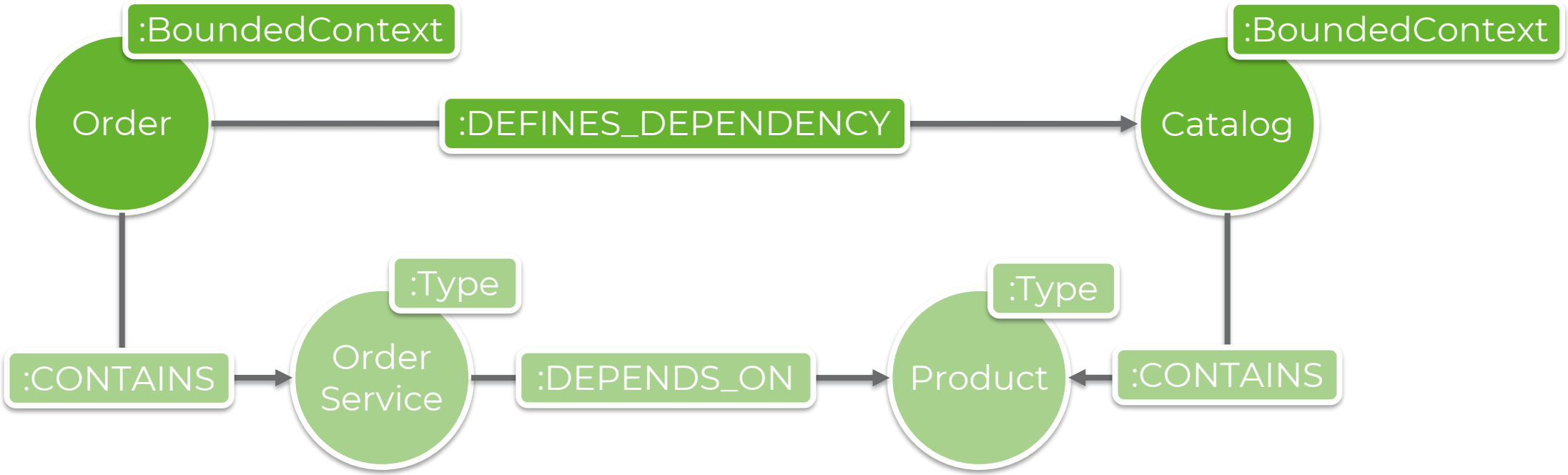
- Existierende Abhängigkeiten zwischen Subdomänen
  - Erlaubte und Unerwartete/Verbotene Abhängigkeiten
  - Anzahl an Abhängigkeiten
  - Hot Spots

SOFTWARE ANALYTICS

- Analyse der fachlichen Strukturierung

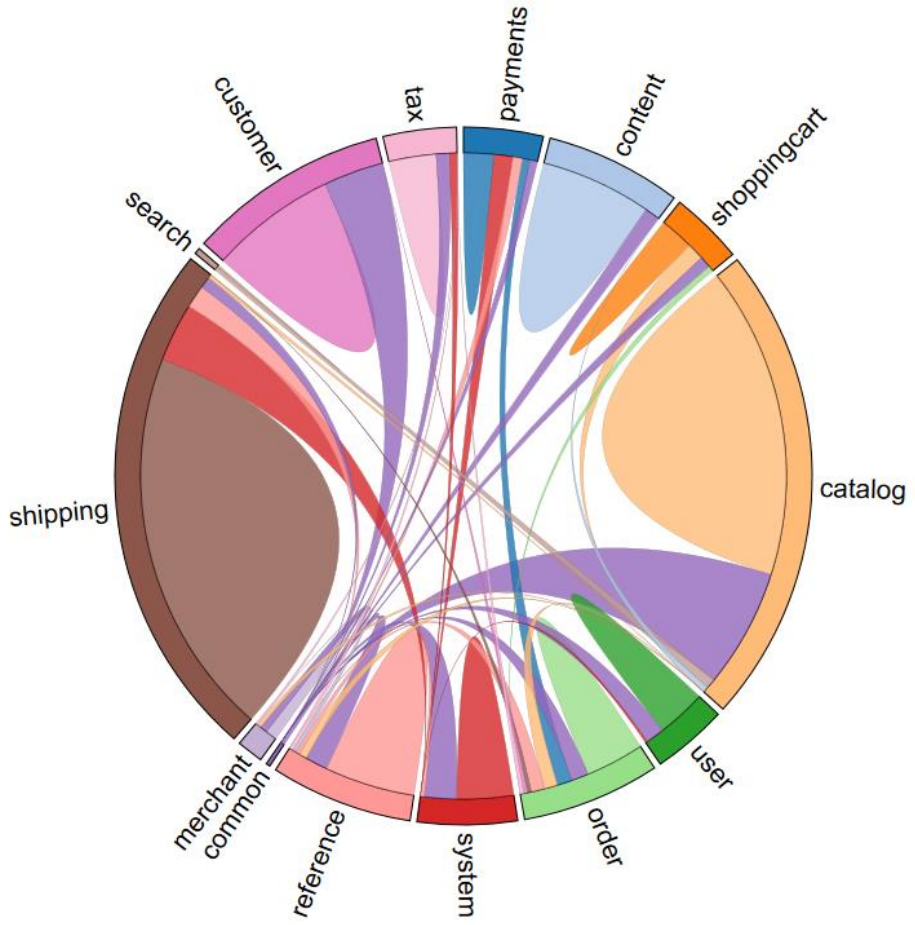
SOFTWARE ANALYTICS

- Analyse der fachlichen Strukturierung



SOFTWARE ANALYTICS

- Live-Demo
  - Visualisierung von Abhängigkeiten
  - Analyse von
    - Ein- und Ausgehenden Abhängigkeiten
    - Kopplungsgrad
    - Hot-Spot Klassen



SOFTWARE ANALYTICS

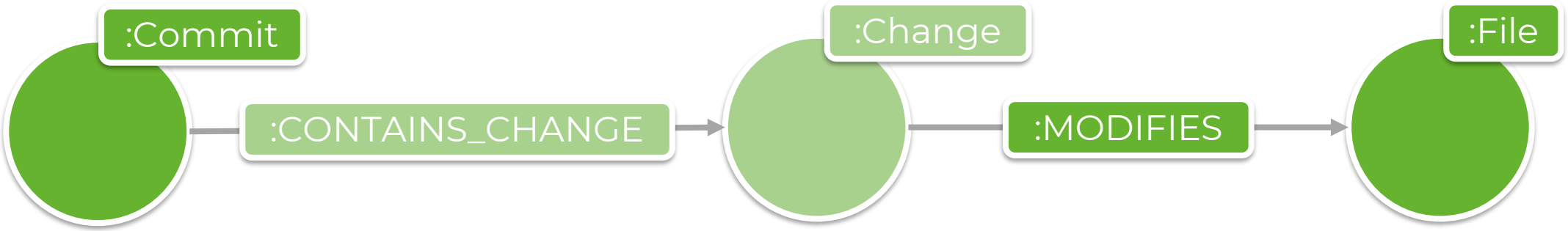
Was sind änderungskritische Anwendungsteile?

SOFTWARE ANALYTICS

- Analyse von häufig geändertem Code
  - Identifikation von wartungsintensiven Hot Spots
  - Auffinden von fehleranfälligem Code
  - Aufdecken von Technischen Schulden
  - Finden von schlecht abgegrenzten Verantwortlichkeiten

SOFTWARE ANALYTICS

- Analyse von häufig geändertem Code



SOFTWARE ANALYTICS

- Analyse von häufig geändertem Code
  - Tiefergehende Analyse von Hot Spots
  - Gezielte Refaktorisierung oder Neuimplementierung
  - Gefahren erkennen, kommunizieren und behandeln

SOFTWARE ANALYTICS

- Live Demo
  - Analyse der Anzahl der Commits je Klasse
    - Fehleranfälliger Code
    - Code ohne klare Verantwortlichkeit
    - Adapter zu externen Systemen

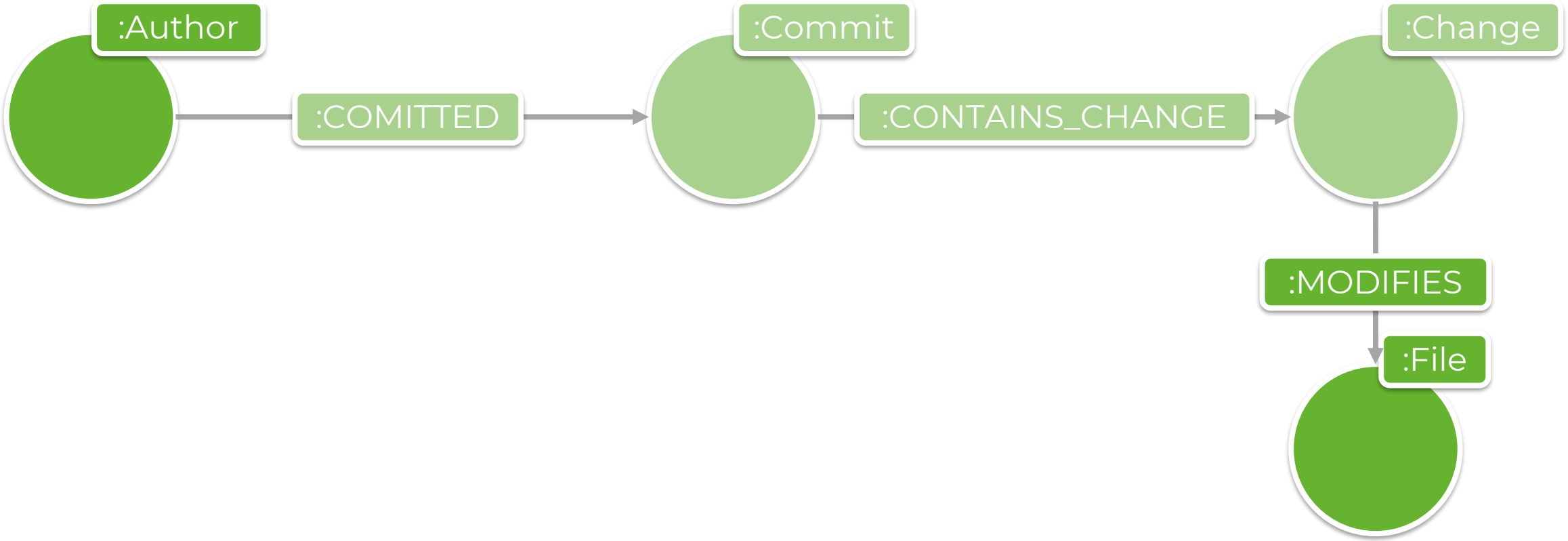
Klasse	Commits
com/salesmanager/shop/filter/StoreFilter.java	88
com/salesmanager/core/business/services/catalog/ProductInfoService.java	78
com/salesmanager/shop/store/controller/shoppingCart/facade/ShoppingCartFacadeImpl.java	76
com/salesmanager/catalog/api/impl/ProductApiImpl.java	72
com/salesmanager/shop/store/controller/order/facade/OrderFacadeImpl.java	70
com/salesmanager/catalog/api/ProductApi.java	69
com/salesmanager/shop/populator/order/ReadableOrderProductPopulator.java	61

SOFTWARE ANALYTICS

- Analyse der Code-Ownership
  - Definition von fachlichen und technischen Ansprechpartnern
  - Dokumentation des vorhandenen Wissens
  - Aufbau von Wissen in unbekannten Bereichen
    - Verstehen und dokumentieren, ggf. Neuimplementierung

SOFTWARE ANALYTICS

- Analyse der Code-Ownership



SOFTWARE ANALYTICS

Und jetzt?

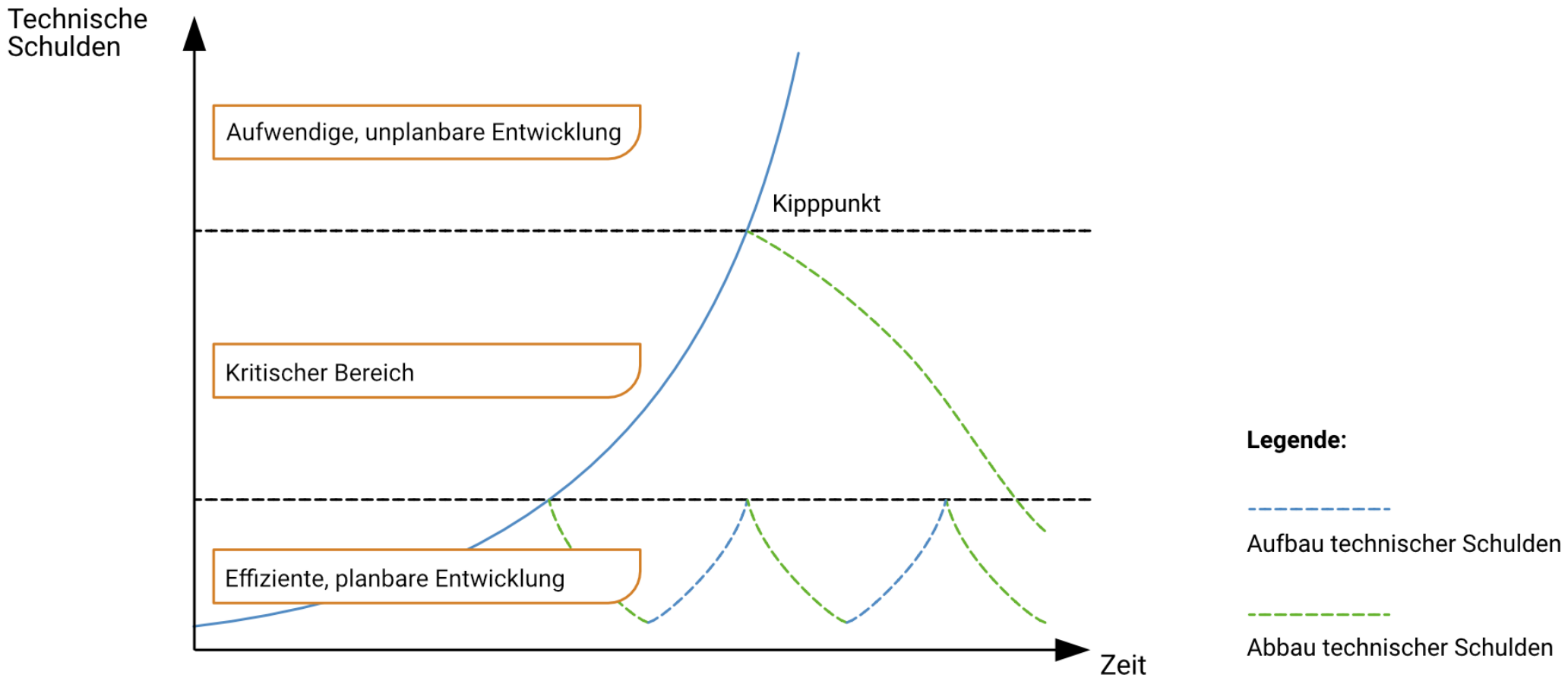
SOFTWARE ANALYTICS

- Bewertung der Ergebnisse
  - Tiefergehende Analysen für kritische Stellen
  - Relevanz und Priorisierung

SOFTWARE ANALYTICS

- Ableitung von Aktionen
  - Refaktorisierung und Restrukturierung
  - Absicherung von (geschaffenen) Strukturen
  - Dokumentation von (Architektur-) Aspekten
  - Management von Technical Debt

SOFTWARE ANALYTICS



SOFTWARE ARCHITECTURE SECURING

SOFTWARE ARCHITECTURE SECURING

- Bewahrung existierender Strukturen vor (erneuter) Erosion
  - Dokumentation des Soll-Zustands
  - Kontinuierliche Analyse des Ist-Zustands
  - Abgleich von Ist- und Soll-Zustand

SOFTWARE ARCHITECTURE SECURING

- Manuelle Pflege und Überprüfung erzeugt hohen Aufwand
  - Dokumentation ist häufig nicht up-to-date
  - Architekturverletzungen häufen sich über die Zeit
  - Kontinuirliche Verschlechterung der Situation

SOFTWARE ARCHITECTURE SECURING

- Lebendige Dokumentation
  - Anwendungsarchitektur basierend auf der Projektstruktur
- Kontinuierliches Reporting
  - Strukturen und Metriken
  - Technische Schulden
  - Architekturverletzungen
  - Programmierrichtlinien

SOFTWARE ARCHITECTURE SECURING

- Dokumentation und Absicherung mit jQAssistant
  - Unmittelbares Feedback an Entwickler
    - Abbruch des CI-Builds bei erkannten Verletzungen
  - Generierung der Dokumentation im Build-Prozess

SOFTWARE ARCHITECTURE SECURING

- Dokumentation und Absicherung mit jQAssistant
  - Concept: Anreichern des Graphen um Abstraktionen
  - Constraint: Auffinden von Verletzungen
  - Group: Ausführung von Konzepten und Constraints

SOFTWARE ARCHITECTURE SECURING

- Formulierung von Regeln in Asciidoc-Dokumenten
  - Nutzung von Cypher
  - Ausgabe als HTML-Dokumente im Build-Prozess
  - Inklusive eingebetteter Ergebnisse (Tabellen und Diagramme)

SOFTWARE ARCHITECTURE SECURING

```
<plugin>
  <groupId>com.buschmais.jqassistant</groupId>
  <artifactId>jqassistant-maven-plugin</artifactId>
  <dependencies>
    <dependency>
      <groupId>org.jqassistant.contrib.plugin</groupId>
      <artifactId>jqassistant-asciidoc-report-plugin</artifactId>
      <version>1.7.0</version>
    </dependency>
  </dependencies>
</plugin>
```

SOFTWARE ARCHITECTURE SECURING

- Entwickler entwickeln ein menthales Modell der Anwendung
  - Bekannte Konzepte in der Anwendung
  - Abbildung von Konzepten im Source Code
  - Annahmen nach eigenem Verständnis und gesammelter Erfahrung

SOFTWARE ARCHITECTURE SECURING

- Hinter jeder Anwendung liegt eine Mustersprache
  - DDD: Bounded Context, Domain Event, Service, …
  - MVC: Model, View, Controller
  - Individuell definierte Mustersprache

SOFTWARE ARCHITECTURE SECURING

- Konzepte der Mustersprache → Definition von Contraints
  - Erlaubte Abhängigkeiten zwischen unterschiedlichen Konzepten
  - Verortung der Konzepte in der Anwendung (bestimmte Packages, …)
- Konzepte der Mustersprache sind im Source Code auffindbar
  - Constraints lassen sich auf Source Code übertragen

SOFTWARE ARCHITECTURE SECURING

- Live-Demo
  - Identifikation von Datentransferobjekten (DTOs)

Concepts

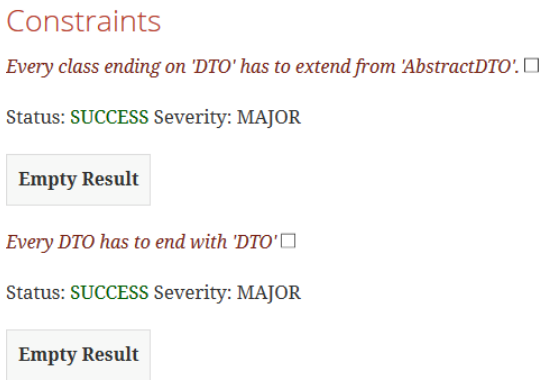
Every non-abstract classes extending from 'AbstractDTO' is marked as a DTO. ☐

Status: SUCCESS Severity: MINOR

Artifact	DTOs
sm-catalog	com.salesmanager.catalog.api.dto.product.ProductImageDTO com.salesmanager.catalog.api.dto.product.DimensionDTO com.salesmanager.catalog.api.dto.product.FinalPriceDTO com.salesmanager.catalog.api.dto.product.ProductOptionValueDTO$ProductOptionValueDescriptionDTO com.salesmanager.catalog.api.dto.product.ProductPriceDTO com.salesmanager.catalog.api.dto.product.ProductDTO com.salesmanager.catalog.api.dto.product.ProductOptionValueDTO com.salesmanager.catalog.api.dto.product.ProductAttributeDTO com.salesmanager.catalog.api.dto.product.ProductDescriptionDTO com.salesmanager.catalog.api.dto.product.AvailabilityInformationDTO com.salesmanager.catalog.api.dto.product.ProductOptionDTO$ProductOptionDescriptionDTO com.salesmanager.catalog.api.dto.product.ProductOptionDTO
sm-core-model-api	com.salesmanager.core.integration.merchant.MerchantStoreDTO com.salesmanager.core.integration.customer.CustomerDTO com.salesmanager.core.integration.tax.TaxClassDTO com.salesmanager.core.integration.language.LanguageDTO

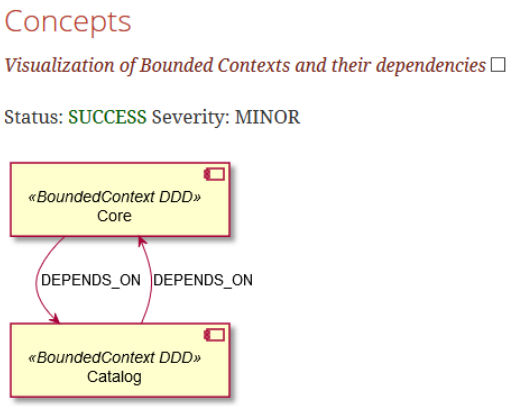
SOFTWARE ARCHITECTURE SECURING

- Live-Demo
  - Absicherung von Vererbungsbeziehungen und Namenskonventionen



SOFTWARE ARCHITECTURE SECURING

- Live-Demo
  - Visualisierung von Abhängigkeiten zwischen Bounded Contexts



ZUM SCHLUSS

ZUM SCHLUSS

- Software Analytics ist ein wertvolles Tool
  - zur Unterstützung während der Entwicklung
  - zur Identifikation von Problemstellen
  - zur Planung von Refaktorisierungen

ZUM SCHLUSS

- Software (Architektur) Dokumentation ist Änderungen ausgesetzt
  - Lebendige Dokumentation hilft, den Änderungsaufwand zu minimieren
  - jQAssisant Konzepte unterstützen die Abbildung im Code und ermöglichen die Erstellung von Dokumentation whrend des Builds

ZUM SCHLUSS

- Erosion der Architektur schreitet kontinuierlich voran
  - Definition von Architekturconstraints macht Regeln für Entwickler sichtbar
  - jQAssistant ermöglicht die Validierung der Constraints während des Builds

VIELEN DANK!

BUSCHMAIS

KONTAKT

Stephan Pirnbaum

stephan.pirnbaum@buschmais.com

Tel. +49 351 320923-22

Twitter: @spirnbaum

BUSCHMAIS GbR

Leipziger Straße 93

01127 Dresden

Tel. +49 351 3209230

info@buschmais.com

www.buschmais.de

ANHANG

```
// Markieren von Projektdateien
MATCH (artifact:Main:Artifact{group: "com.shopizer"})
SET artifact:Shopizer
WITH artifact
MATCH (artifact)-[:CONTAINS*]->(c)
SET c:Shopizer
RETURN artifact.name AS Artifact,
       count(DISTINCT c) AS Content_Cnt
ORDER BY artifact.name
```

CYPHER-ABFRAGEN

Technische Strukturierung

```
// Abhängigkeiten zwischen Artefakten
MATCH (a1:Artifact:Shopizer)-[:CONTAINS*]->(t1:Type:Shopizer),
      (a2:Artifact:Shopizer)-[:CONTAINS*]->(t2:Type:Shopizer),
      (t1)-[dep:DEPENDS_ON]->(t2)
WHERE a1 <> a2
RETURN a1.name AS Source,
       a2.name AS Target,
       COUNT(DISTINCT dep) AS Dependencies
ORDER BY a1.name
```

```
// Anlegen eines Knoten je Layer
MERGE (dataLayer:Layer {name:"DataLayer"})
MERGE (domainLayer:Layer {name:"DomainLayer"})
MERGE (presentationLayer:Layer {name:"PresentationLayer"})
```

```
// Zuordnung von Artefakten zu Schichten
MATCH (a:Artifact:Shopizer)-[:CONTAINS*]->(t)
WITH DISTINCT t,
     CASE a.name
         WHEN "sm-shop" THEN "PresentationLayer"
         WHEN "sm-core" THEN "DomainLayer"
         WHEN "sm-core-modules" THEN "DomainLayer"
         WHEN "sm-core-model" THEN "DataLayer"
     END AS layer
MATCH (l:Layer {name:layer})
MERGE (l)-[:CONTAINS]->(t)
RETURN l.name AS Layer, count(DISTINCT t) AS Size
ORDER BY Size DESC
```

```
// Propagieren der Abhängigkeiten auf Schichten
MATCH (l1:Layer)-[:CONTAINS]->(t1:Type),
      (l2:Layer)-[:CONTAINS]->(t2:Type),
      (t1)-[:DEPENDS_ON]->(t2)
WHERE l1 <> l2
WITH DISTINCT l1, l2
MERGE (l1)-[:DEPENDS_ON]->(l2)
RETURN l1.name AS Dependent, l2.name AS Dependency
```

CYPHER-ABFRAGEN

Fachliche Strukturierung

```
// Anlegen eines Knoten je Fachlichkeit
MATCH (p:Package:Shopizer)-[:CONTAINS]->(subDomain:Package:Shopizer)
WHERE p.fqn = "com.salesmanager.core.business.services"
WITH collect(DISTINCT subDomain.name) AS subDomains
UNWIND subDomains AS subDomain
MERGE (sD:SubDomain {name: subDomain})
```

```
// Zuordnung der Klassen zur fachlichen Domäne
MATCH (sD:SubDomain),
      (p:Package:Shopizer)-[:CONTAINS*]->(t:Type:Shopizer)
WHERE p.name = sD.name
MERGE (sD)-[:CONTAINS]->(t)
RETURN sD.name AS  SubDomain,
       count(DISTINCT t) AS Classes
```

```
// Prozentualer Anteil an zugeordnodeten Klassen
MATCH (t:Type:Shopizer)
WITH count(t) AS Total
MATCH (:SubDomain)-[:CONTAINS]->(t:Type:Shopizer)
RETURN 100 * count(t) / Total AS Coverage
```

```
// Abhängigkeiten zwischen Subdomänen, Class-Level
MATCH (artifact:Main:Artifact{group: "com.shopizer"})
SET artifact:Shopizer
WITH artifact
MATCH (artifact)-[:CONTAINS*]->(c)
SET c:Shopizer
RETURN artifact.name AS Artifact,
       count(DISTINCT c) AS Content_Cnt
ORDER BY artifact.name
```

```
// Abhängigkeiten zwischen Subdomänen, Line-Level
MATCH (sD1:SubDomain {name: "catalog"})-[:CONTAINS]->(t1:Type:Shopizer),
      (sD2:SubDomain)-[:CONTAINS]->(t2:Type:Shopizer),
      (t1)-[dep:DEPENDS_ON]->(t2)
WHERE sD2.name <> "catalog"
RETURN sD1.name AS Source,
       sD2.name AS Target,
       sum(dep.weight) AS Dependencies
ORDER BY Dependencies DESC
```

```
// Ausgehende Abhängigkeiten des Katalogs, Hot Spots
MATCH (sD1:SubDomain {name: "catalog"})-[:CONTAINS]->(t1:Type:Shopizer),
      (sD2:SubDomain)-[:CONTAINS]->(t2:Type:Shopizer),
      (t1)-[dep:DEPENDS_ON]->(t2)
WHERE sD2.name <> "catalog"
RETURN t2.fqn AS Dependency,
       sum(dep.weight) AS Dependencies
ORDER BY Dependencies DESC
LIMIT 5
```

CYPHER-ABFRAGEN

Git-Analyse

```
// Häufig geänderter Code
MATCH (c:Git:Commit)-[:CONTAINS_CHANGE]->(:Change)-[:MODIFIES]->(f:File{type: "java"})
WHERE NOT c:Merge
RETURN f.fileName AS Files,  count(c) AS Changes
ORDER BY Changes DESC
LIMIT 30
```