

SOFTWAREMODERNISIERUNG? PRAGMATISCH, PRAKTISCH, ERFOLGREICH MIT ARCHITECTURE DECISION RECORDS

Stephan Pirnbaum @spirnbaum

techcamp Legacy & Innovation, Hamburg

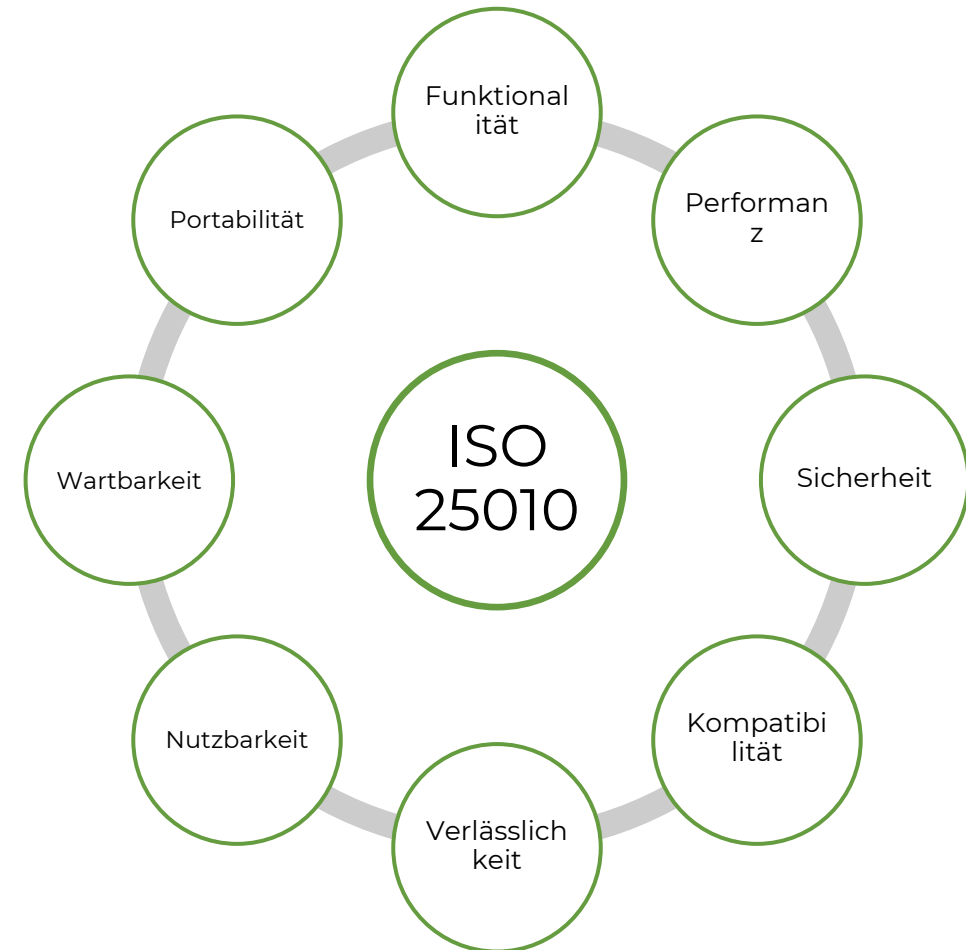
19.06.2024

- ^ Wer wir sind
 - ^ Dresdner IT-Beratungsunternehmen, gegründet im Jahre 2008
- ^ Unsere Schwerpunkte
 - ^ Architekturberatung und Entwicklung moderner Geschäftsanwendungen
 - ^ Software-Qualitätsanalysen und -Sicherung
 - ^ Köpfe hinter jQAssistant: Entwicklung, Trainings, Workshops
- ^ Unsere Kunden im Bereich Software-Qualitätsanalyse
 - ^ u. a. ITZBund, Sächsische Aufbaubank-Förderbank, ASML, GlobalFoundries, Thyssenkrupp Steel

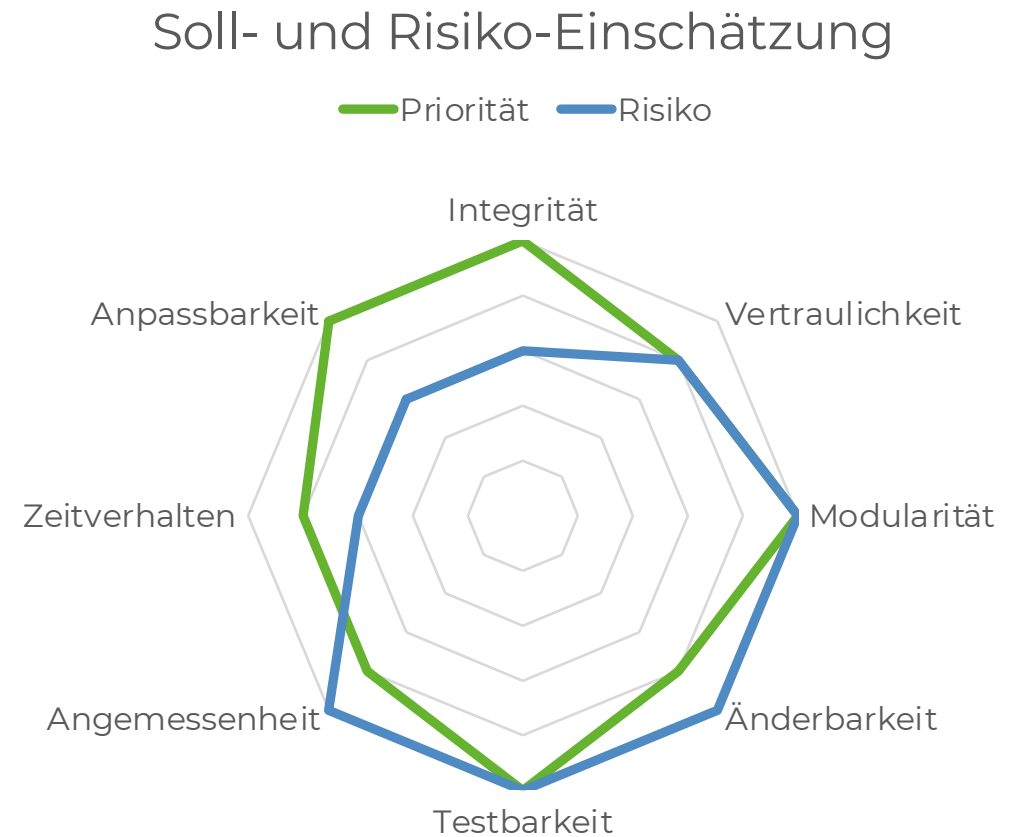
ARCHITEKTUR

Einmal super – Immer super?

Architektur ist der **Enabler**
zur Erreichung von
Qualitätszielen



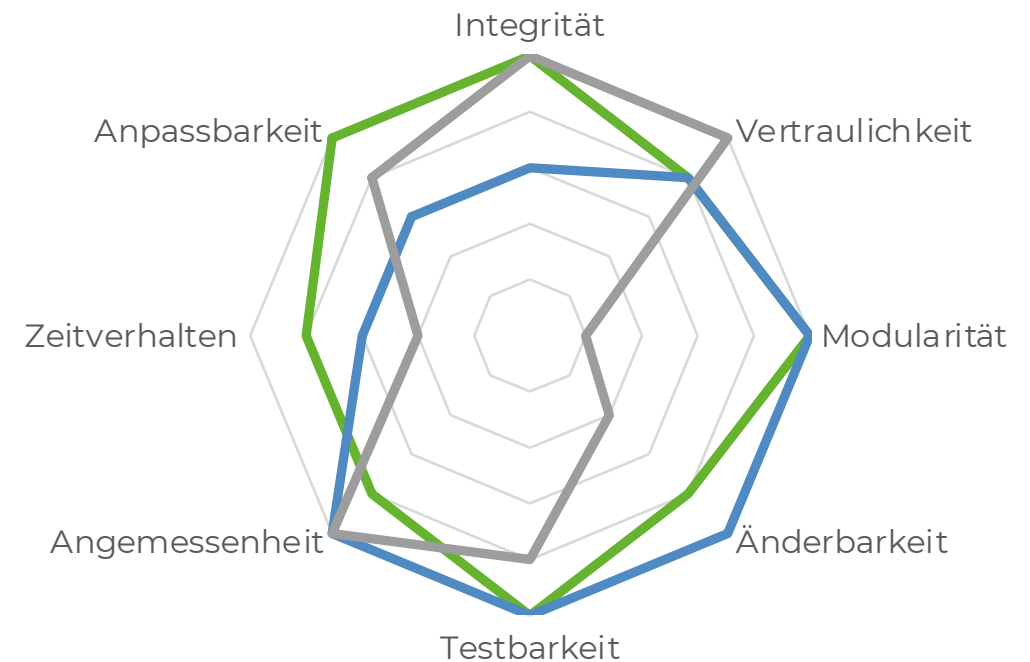
Architektur wird auf **Basis**
gesetzter Qualitätsziele
entworfen



Anforderungen und verfügbare Technologien **ändern** sich, Design und Implementierung laufen **auseinander**

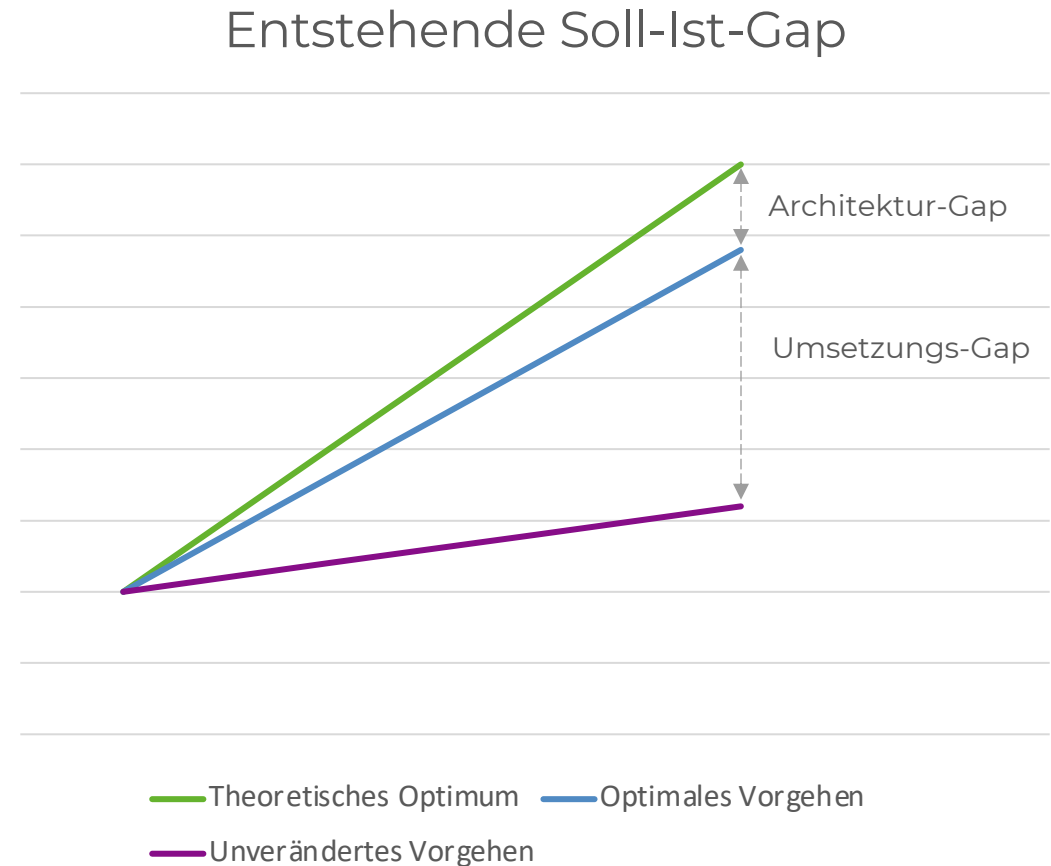
Soll-, Ist- und Risiko-Einschätzung

— Priorität — Risiko — Erfüllungsgrad



▲ Abweichungen durch

1. Diskrepanz zwischen Anforderungen und Architektur
2. Auseinanderlaufen von Architektur und Implementierung

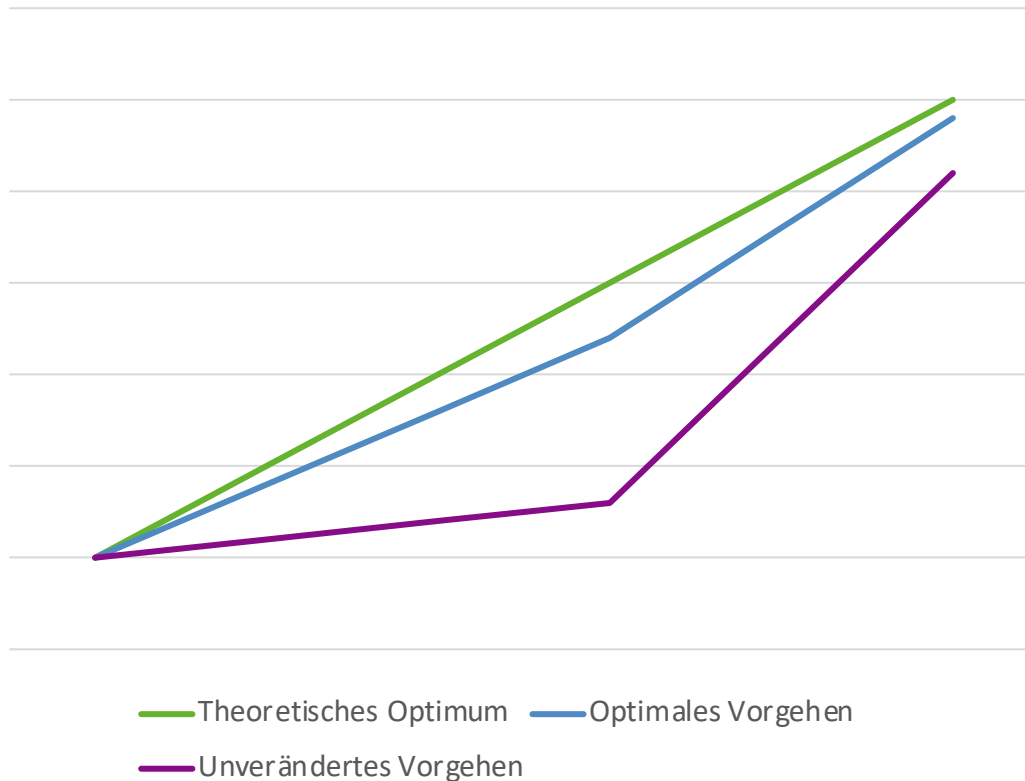


Entstehung von
Diskrepanzen ist ein
natürlicher Prozess

Entstehung von
Diskrepanzen ist ein
natürlicher Prozess

... welcher **abgeschwächt**
werden kann und muss

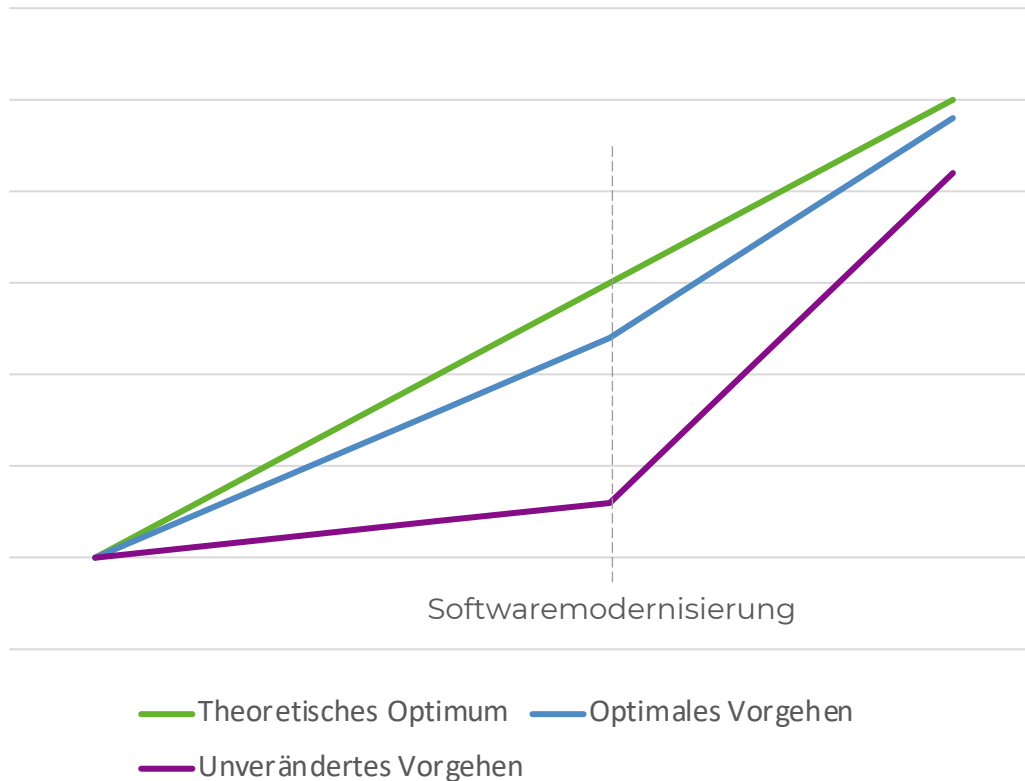
Mitigation der Soll-Ist-Gap



Λ Lösung erfolgt durch

1. Anpassung der Architektur durch neue Entscheidungen
2. Strikter Abgleich von Architektur und Umsetzung

Mitigation der Soll-Ist-Gap



Λ Lösung erfolgt durch

1. Anpassung der Architektur durch neue Entscheidungen
2. Strikter Abgleich von Architektur und Umsetzung

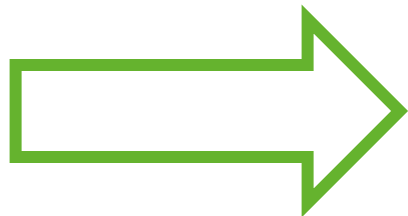
SOFTWAREMODERNISIERUNG

▲ Modernisierungsdruck durch ext. und int. Faktoren

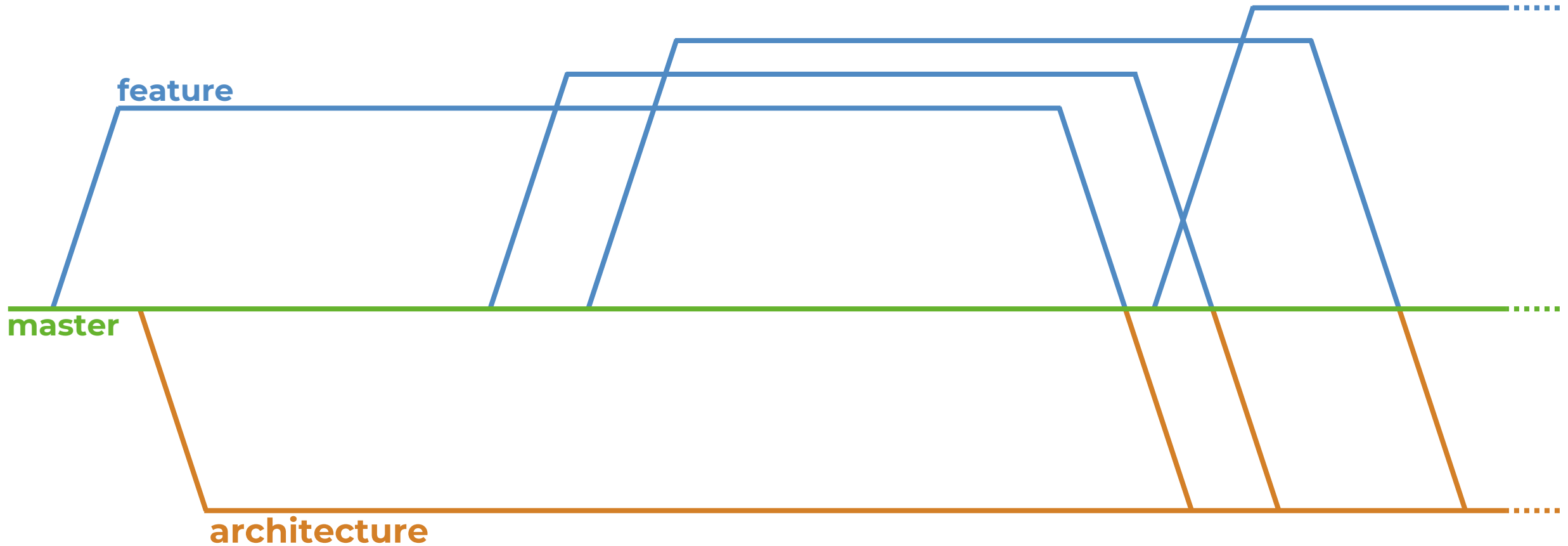
- ▲ Neue Anforderungen
- ▲ Neue Technologien
- ▲ Neue Konkurrenten
- ▲ Kostenoptimierung
- ▲ Prozessoptimierung

▲ Kontinuierliche Modernisierung der Architektur notwendig

ABER: Funktionale Weiterentwicklung kann nicht
ausgesetzt werden



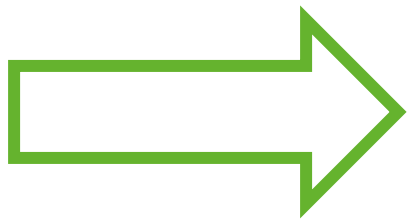
Parallele Modernisierung



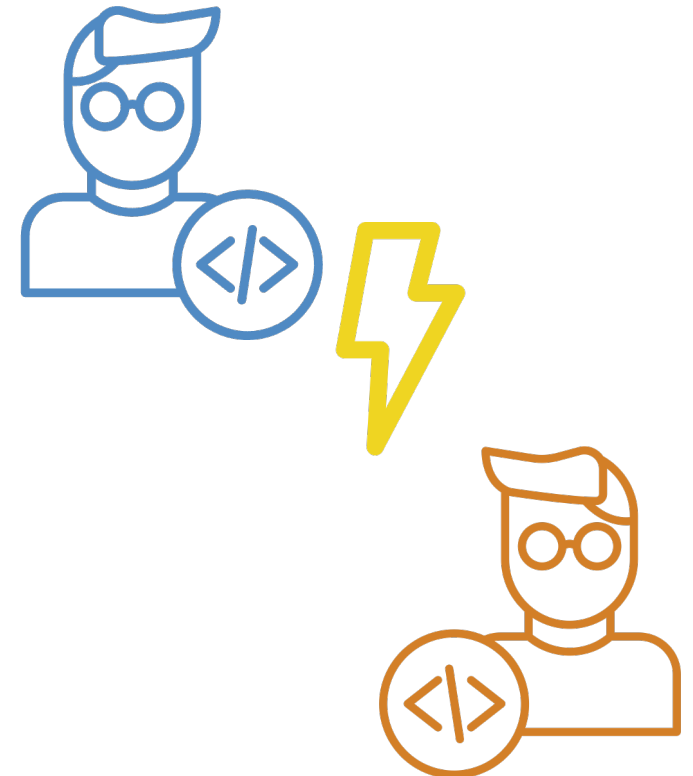
Ein typischer Projektverlauf

▲ Parallele Entwicklung an Features und Architektur

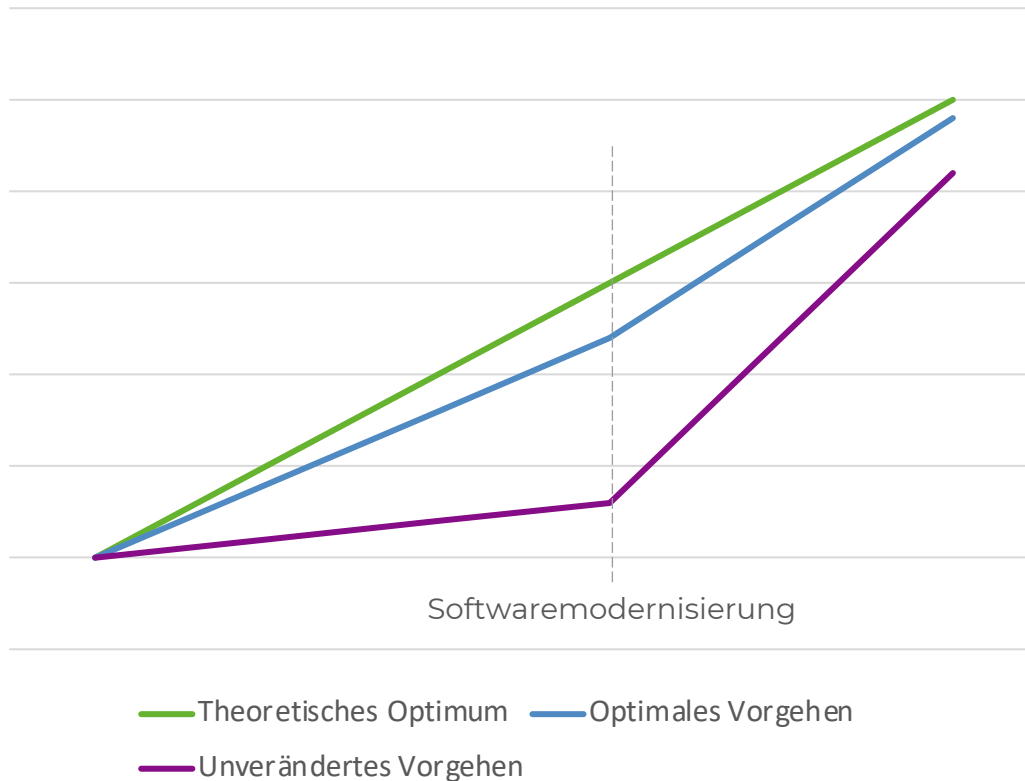
- ▲ Entwickler arbeiten gegeneinander
- ▲ Steigender Integrationsaufwand



Endlose Geschichte



Mitigation der Soll-Ist-Gap



^ Erfolgskriterien

1. Effiziente Dokumentation der Entscheidungen
2. Effiziente Validierung der Umsetzung

AD WAS?

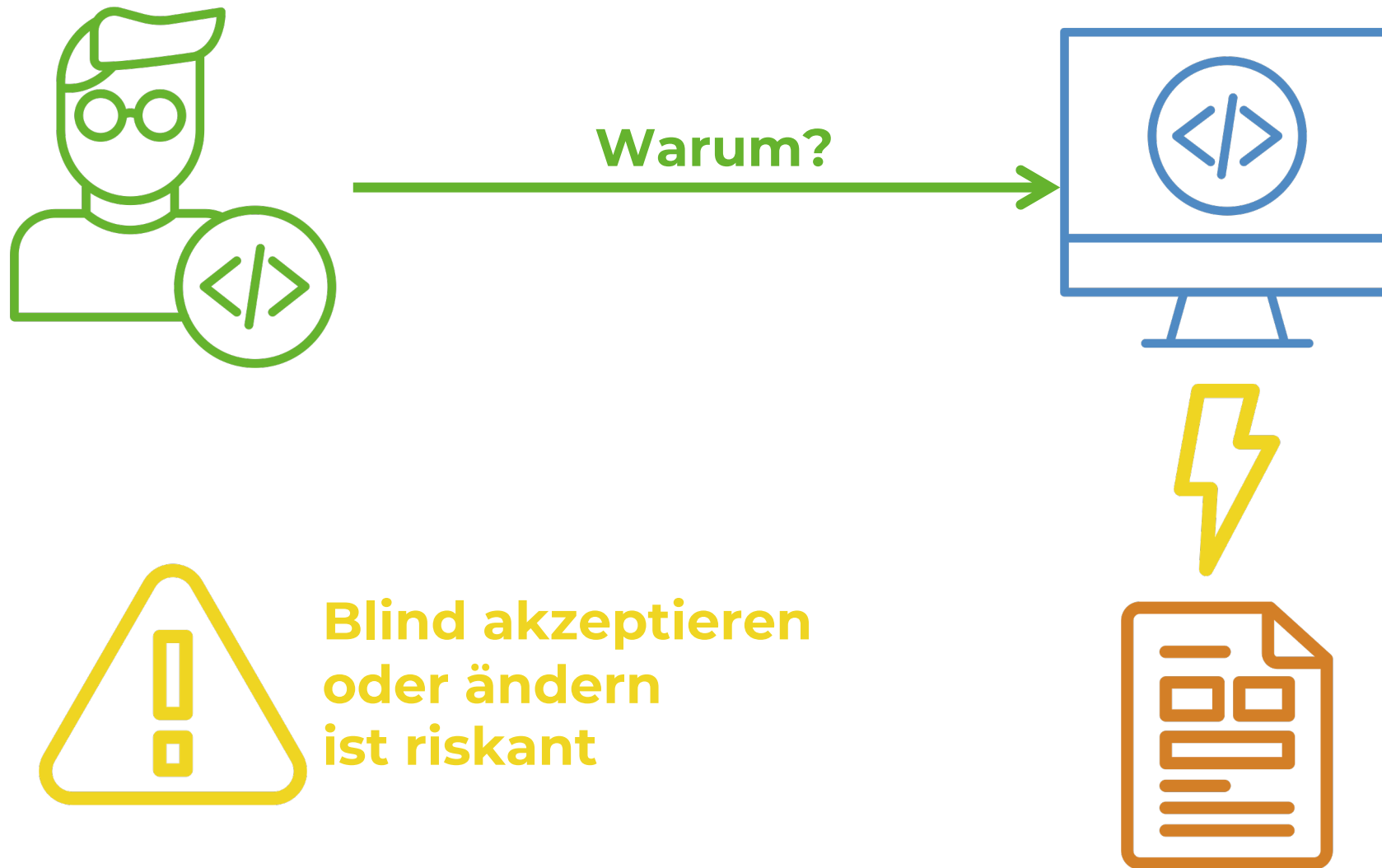
^ Michael Nygard (2011)

“Documents that assist the team itself can have value, but only if they are kept up to date.”

“Large documents are never kept up to date.”

“Nobody ever reads large documents, either.”





- ^ Dokumentation muss
 - ^ Entscheidungen explizit festhalten
 - ^ Kontext der Entscheidung darlegen
 - ^ Auswirkungen auf Entwicklung aufzeigen
 - ^ Leichtgewichtig sein

^ Architecture Decision Records

^ Ein Dokument je Entscheidung

- ✓ Klein
- ✓ Übersichtlich
- ✓ Fokussiert

ADR # - ...

Status: ...

Kontext: ...

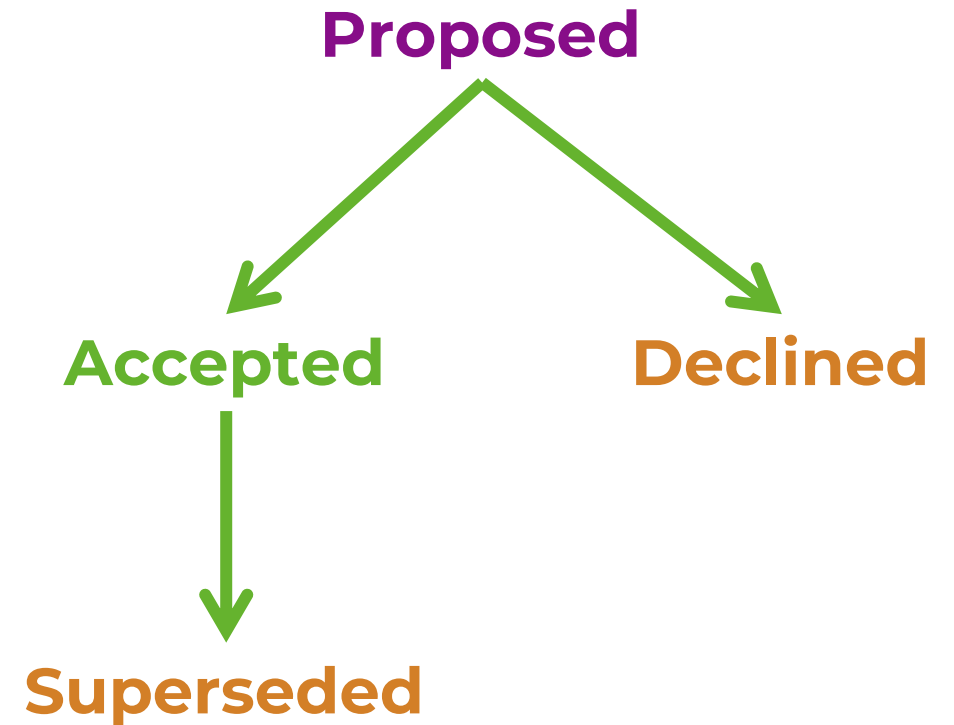
Entscheidung: ...

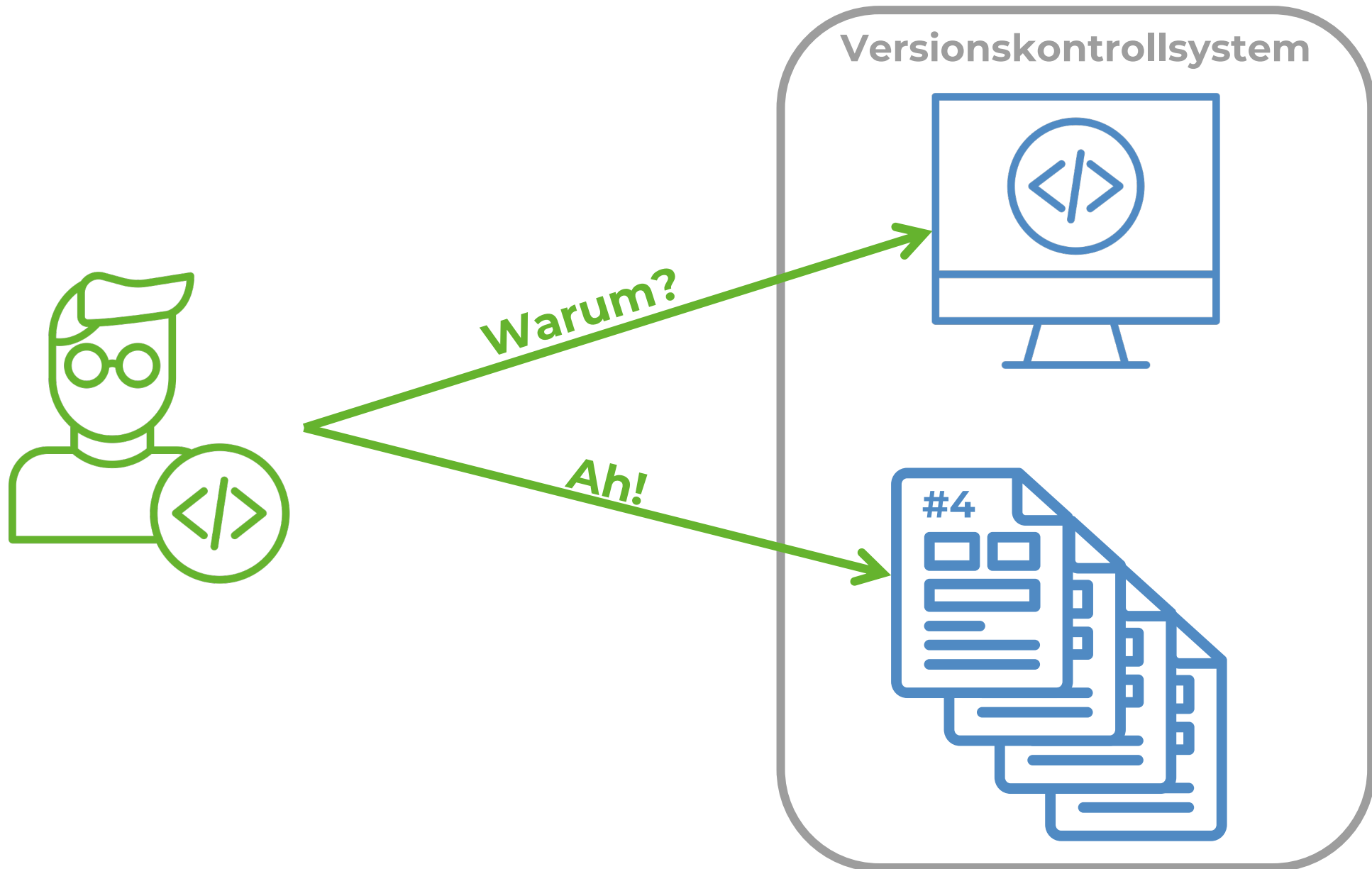
Konsequenzen: ...

^ Architecture Decision Records

^ Version mit Status

- ✓ Nachvollziehbar
- ✓ Relevant
- ✓ Aktuell





ADRS MODERNISIERUNG

Ein Beispiel

^ Online-Shop

- ^ Produktkatalog

- ^ Warenkorb

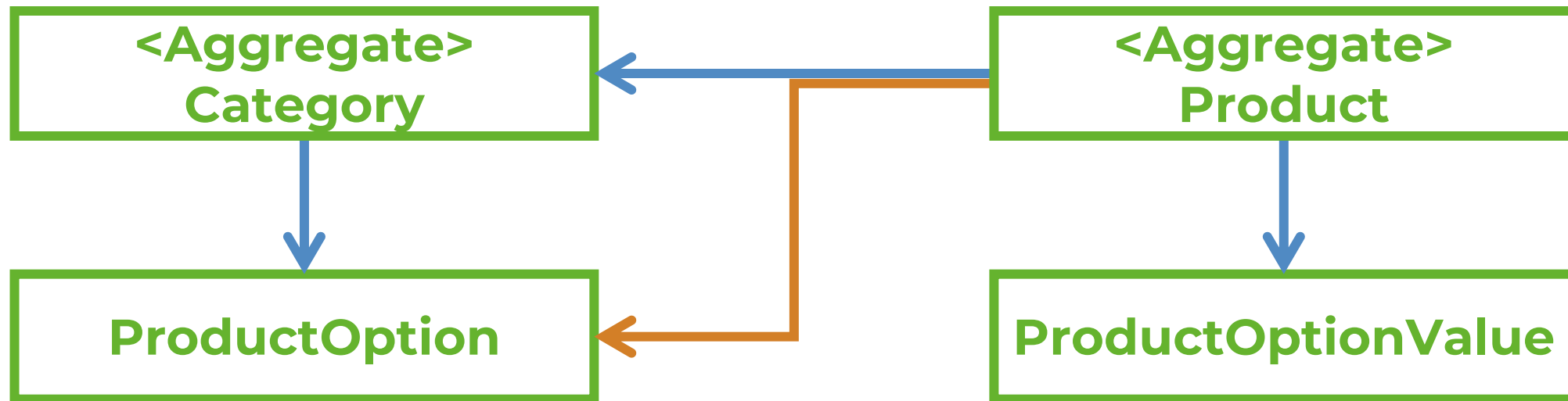
- ^ Nutzerkonten

- ^ Bestellabwicklung

- ^ ...

- ▲ Bevorstehende Änderung der DB für den Produktkatalog
 - ▲ (Aufgrund Performanz, Pflege, ...)
 - ▲ Relationale DB → Dokumentenbasierte DB
 - ▲ Evaluation und Proof of Concept ist bereits erfolgreich umgesetzt
 - ▲ Shop soll entsprechend vorbereitet werden

- ^ Dokumentenbasierte DB erfordert Aggregate im Modell
- ^ Deep Linking zwischen Aggregaten ist verboten



▲ Herausforderungen

- ▲ Weiterentwicklung vs. Umsetzung der Architekturentscheidung
- ▲ Planbarkeit der Umsetzung → Aufwand?
- ▲ Monitoring der Fortschritte

ADRS MODERNISIERUNG

Pragmatisch!

^ Dokumentation mit AsciiDoc

- ^ Leichtgewichtig
- ^ Versionierbar
- ^ Verschiedene Ausgabeformate

= 001 – ...

== Status Proposed

== Kontext

* Migration der DB des
Produktkatalogs

== Entscheidung

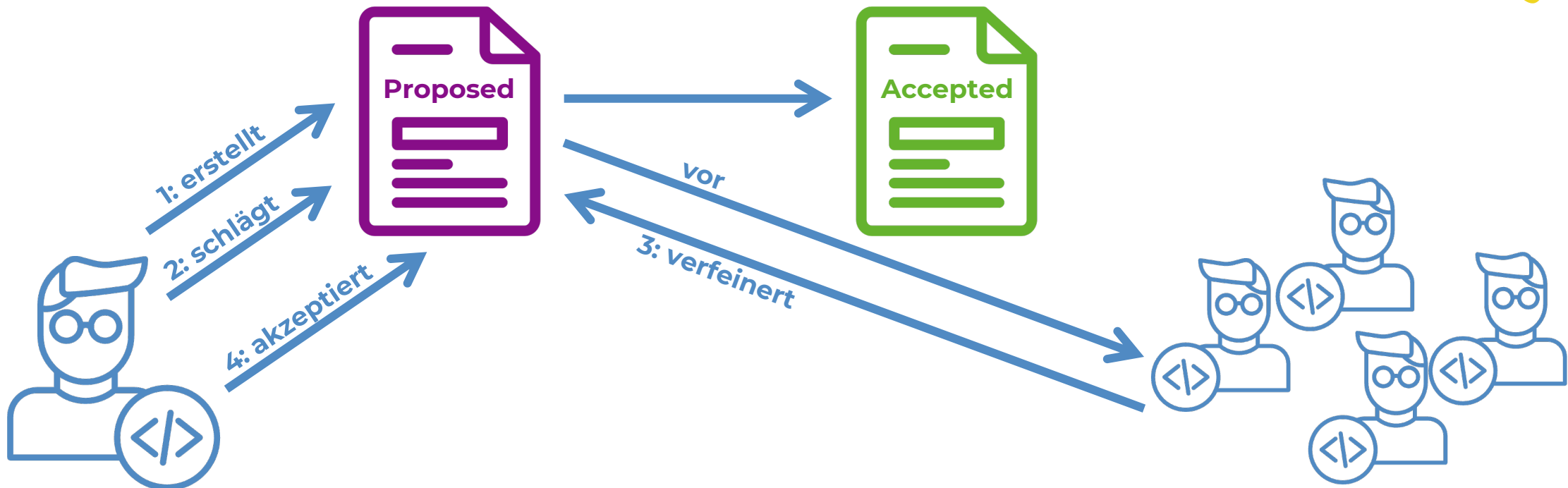
* Aggregatbildung
** Keine Deep Links

== Konsequenzen

* Alle Entitäten nur durch 1
Aggregat referenziert
** Existierende Deep Links
auflösen

- ▲ Prozess möglichst schmal halten
- ▲ Wichtig: Jeden Beteiligten informieren

 Nicht alle im Team müssen an der Entscheidung beteiligt sein



- ▲ Vereinfachung der Arbeit mit ADRs durch
 - ▲ Bereitgestelltes ADR-Template
 - ▲ Thematische Unterordner (Struktur, Persistenz, Logging, ...)
 - ▲ Klarer Prozess von der Idee zum finalen ADR
 - ▲ Gute IDE-Integration (AsciiDoc-Plugin)

ADRS MODERNISIERUNG

Praktisch!

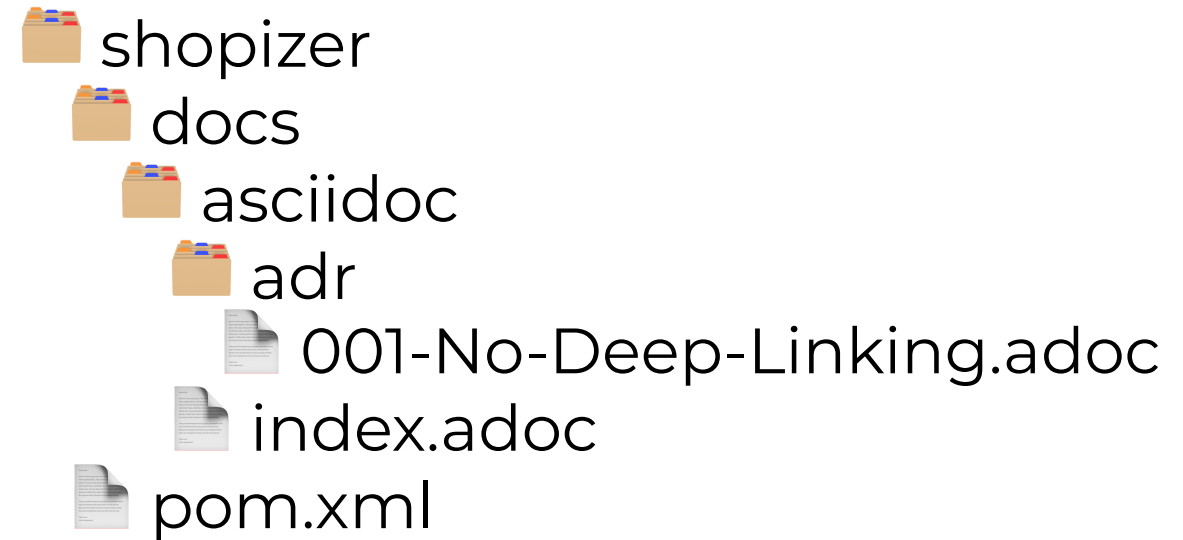
▲ Ablage im Repository

- ▲ Code-nah

- ▲ Versioniert

 - ▲ Nachvollziehbar

- ▲ Integration mit z.B. Maven



^ Rendering der Dokumente

^ während des Builds

^ in verschiedenen Formaten

^ zur Bereitstellung z.B.

^ als Maven Site

^ in Confluence

Table of Contents

Content

Data Preparation

Concepts

Architecture Decision Records

Introduction

Working with ADRs

Template

Creating an ADR

Updating an ADR

Decisions

001 - Use Slf4j for Logging

002 - Don't Log and Throw an Exception

003 - No Deep Links between Aggregates

003 - No Deep Links between Aggregates

Status

Proposed

Context

The upcoming switch from a relational to a document-based database technology for the catalog sub-domain requires the implementation of aggregates in the data model.

While those aggregates can be identified manually, it's required that there are no deep links between different aggregates.

An entity linked to by two aggregates as depicted in the figure would thus violate the requirement.

Decision

Each entity must only be referenced in one aggregate, i.e. by one aggregate root and not by entities from outside the aggregate.

Following constructs must therefore be avoided:

- An Entity which is referenced by two or more Aggregate Roots
- An Entity which is referenced by an Entity which is not part of the same Aggregate

For easier identification of classes that represent an Aggregate Root, they will be annotated with `org.jqassistant.contrib.plugin.ddd.annotation`.

Consequences

- Existing Code
 - All Aggregate Roots in the data model of the catalog sub-domain (`com.salesmanager.catalog`) must be identified and marked as such
 - All violations must be identified and fixed
- New Code
 - If a new Aggregate Root is introduced, it needs to be marked as such
 - Deep Links from one Aggregate to another must not be added. Aggregates may only reference each other via Aggregate Roots.

▲ ADR ist nah am Code, ABER:

▲ Wer prüft die Aktualität und Relevanz der Regeln?

▲ Wer weist den Entwickler auf Regelverletzungen hin?

▲ Wer überprüft Fortschritte und zeigt verbleibende Aufwände?

jQAAssistant

Your Software. Your Structures. Your Rules.



scan

+



document

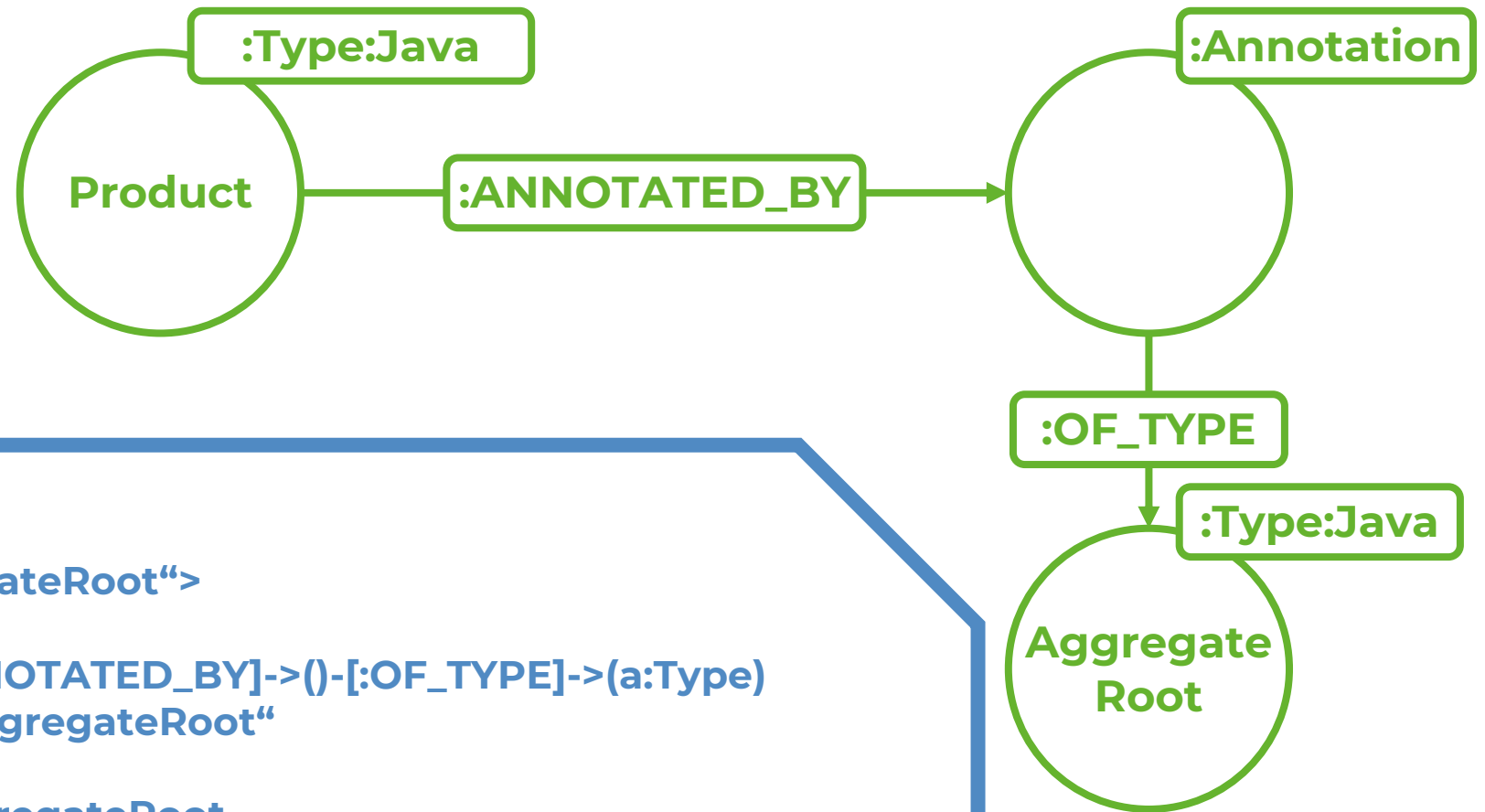
=



validate

- ▲ Integration mit jQAssistant ermöglicht
 - ▲ Identifikation von Architekturkonzepten im Code
 - ▲ z. B. Aggregate, Bounded Contexts
 - ▲ Überprüfung von definierten Architekturregeln
 - ▲ z. B. kein Deep Linking zwischen Aggregaten
 - ▲ Transparenz über Architekturschulden und Migrationsaufwände
 - ▲ Verletzungen als „ToDo-Liste“ für die Entwicklung

^ Konzepte

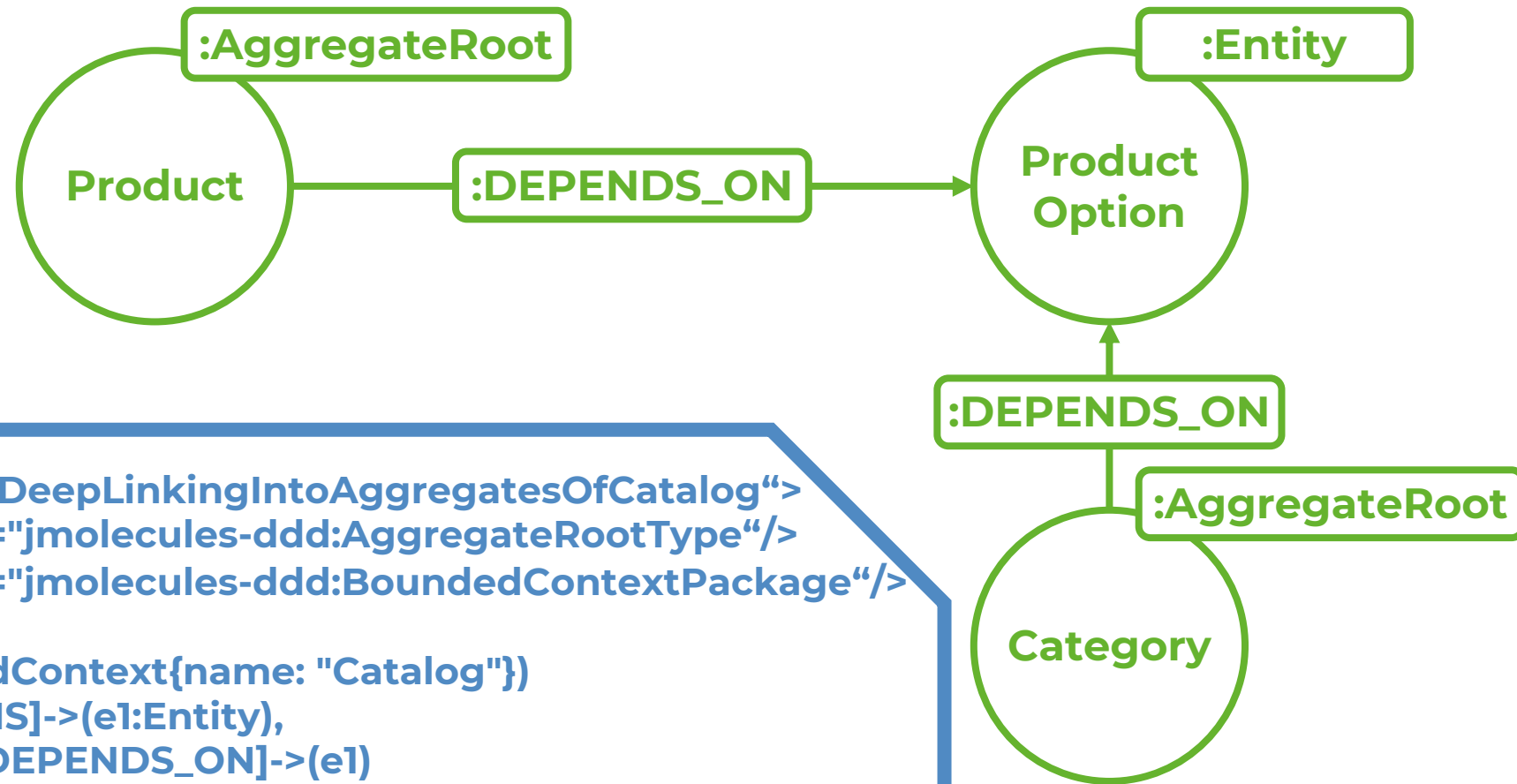


```
<concept id="ddd:AggregateRoot">
  <cypher>
    MATCH (t:Type)-[:ANNOTATED_BY]->()-[:OF_TYPE]->(a:Type)
    WHERE a.name = „AggregateRoot“
    SET t:AggregateRoot
    RETURN t.fqn AS AggregateRoot
  </cypher>
</concept>
```



Es gibt jMolecules und
ein jQA-Plugin dafür 😊

^ Regeln



```
<constraint id="adr:IllegalDeepLinkingIntoAggregatesOfCatalog">
  <requiresConcept refId="jmolecules-ddd:AggregateRootType"/>
  <requiresConcept refId="jmolecules-ddd:BoundedContextPackage"/>
  <cypher>
    MATCH (bC1:BoundedContext{name: "Catalog"})
      -[:CONTAINS]->(e1:Entity),
      (e2:Entity)-[:DEPENDS_ON]->(e1)
    WHERE NOT exists((bC1)-[:CONTAINS]->(e2))
    RETURN e1.fqn AS CatalogDependency,
           e2.fqn AS DependentClass,
           e1:AggregateRoot AS IsAggregateRoot
  </cypher>
</constraint>
```

- ▲ Rendering der Ergebnisse
- ▲ via asciidoctor-maven-plugin
 - ▲ jqassistant-asciidoctorj-extensions zur Integration der Regeln in das entstehende HTML
- ▲ inklusive Query und Status (Erfolgreich/Fehlgeschlagen)
- ▲ auch grafisch (PlantUML)

Table of Contents

Content

Architecture Decision Records

Introduction

Working with ADRs

Template

Creating an ADR

Updating an ADR

Architecture Decision Records

001 - Use Sif4j for Logging

002 - Don't Log and Throw an Exception

003 - No Deep Links between Aggregates

jqAssistant Summary

adr-001-logging:IllegalUsageOfSystemOutAndErr

adr-001-logging:UseSif4jForLogging

adr-002-log-and-throw:IllegalLogAndThrow

adr-003-deep-linking:IllegalDeepLinkingIntoAggregatesOfCatalog

adr-003-deep-linking:IllegalDeepLinkingBetweenAggregates

- Consequences
- Existing Code
 - All Aggregate Roots in the data model of the catalog sub-domain (`com.salesmanager.catalog`) must be identified and marked as such
 - All violations must be identified and fixed
 - New Code
 - If a new Aggregate Root is introduced, it needs to be marked as such
 - Deep Links from one Aggregate to another must not be added. Aggregates may only reference each other via Aggregate Roots.

Id	Description	Severity	Status
adr-003-deep-linking:IllegalDeepLinkingIntoAggregatesOfCatalog	Return all Entities and Aggregate Root objects from the Catalog Bounded Context which are referenced from another context.	MAJOR	FAILURE
adr-003-deep-linking:IllegalDeepLinkingBetweenAggregates	Return all Entities that are part of more than one Aggregate via their Transitive Closure.	MAJOR	SUCCESS

com.salesmanager.shop.store.security.manager.JWTCustomerAuthenticationManager	47
---	----

adr-003-deep-linking:IllegalDeepLinkingIntoAggregatesOfCatalog


ERROR

Return all Entities and Aggregate Root objects from the Catalog Bounded Context which are referenced from another context.
 Status: **FAILURE**, Severity: MAJOR

CatalogDependency	DependentClass
com.salesmanager.catalog.model.product.Product	com.salesmanager.core.model.shoppingcart.ShoppingCart
com.salesmanager.catalog.model.product.review.ProductReview	com.salesmanager.core.model.customer.Customer
com.salesmanager.catalog.model.product.attribute.ProductAttribute	com.salesmanager.core.model.shoppingcart.ShoppingCart

adr-003-deep-linking:IllegalDeepLinkingBetweenAggregates


SUCCESS

Return all Entities that are part of more than one Aggregate via their Transitive Closure.
 Status: **SUCCESS**, Severity: MAJOR

ADRS MODERNISIERUNG

Erfolgreich!

- ▲ Leichte Nachvollziehbarkeit der Entscheidungen
 - ▲ Besonders für neue Entwickler
- ▲ Leichtgewichtige Dokumentation einfach verteilbar
- ▲ Manifestation von Wissen

- ▲ Definition von Architekturkonzepten und -regeln
 - ▲ Kontinuierliches Monitoring der Fortschritte
 - ▲ Genaue Planung der Umsetzung (Abschätzung)
 - ▲ Automatisierte Absicherung der Umsetzung der Entscheidungen

DANKE !

NOCH FRAGEN?

www.buschmais.de

BUSCHMAIS



KONTAKT

Stephan Pirnbaum

stephan.pirnbaum@buschmais.com

Tel. +49 351 320923-22

X: @spirnbaum

LinkedIn: Stephan Pirnbaum

BUSCHMAIS GbR

Leipziger Straße 93

01127 Dresden

Tel. +49 351 3209230

info@buschmais.com

www.buschmais.de