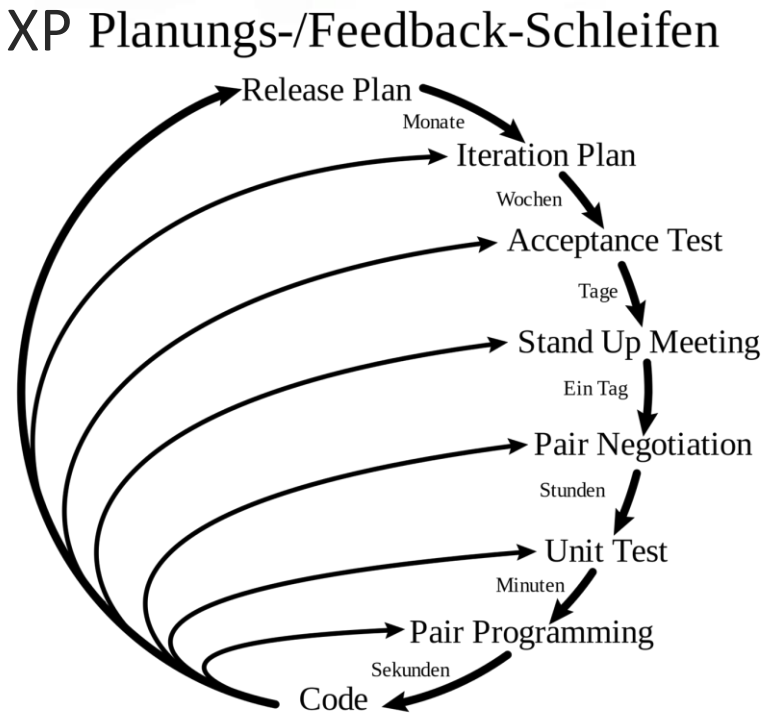
Emergente Testarchitektur

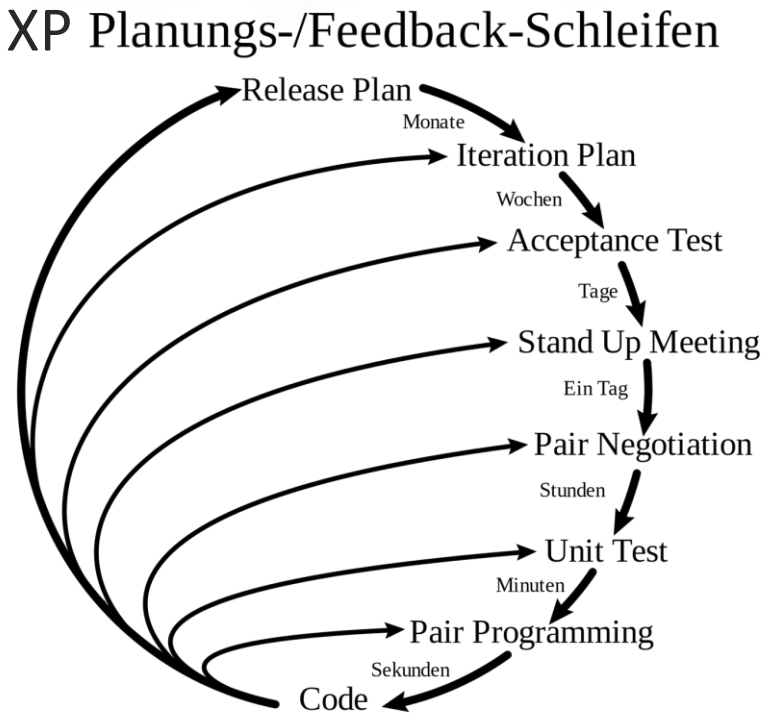
Moritz Tiedje

moritz.tiedje@andrena.de

Warum schreiben wir Tests?



Warum schreiben wir Tests?



Warum?
=> Feedback

Warum schreiben wir Tests?

- Freiheit von hohem manuellen Testaufwand

Warum schreiben wir Tests?

- Freiheit von hohem manuellen Testaufwand
- Lebendige Dokumentation

Warum schreiben wir Tests?

- Freiheit von hohem manuellen Testaufwand
- Lebendige Dokumentation
- Leichte Wartbarkeit

Warum schreiben wir Tests?

- Freiheit von hohem manuellen Testaufwand
- Lebendige Dokumentation
- Leichte Wartbarkeit
- Anforderungsdefinition (TDD)

Warum schreiben wir Tests?

- Freiheit von hohem manuellen Testaufwand
- Lebendige Dokumentation
- Leichte Wartbarkeit
- Anforderungsdefinition (TDD)
- Grundbaustein im Entwicklerworkflow:
  - Write Failing Test -> Make it pass -> Refactor (TDD)
  - Get it working –> Test it –> Refactor

Wie erkenne ich einen guten Test?

Ein guter Test gibt **schnelles**, **wertvolles** und **kosteneffizientes** Feedback zum Verhalten des Programms.

Schnell:

- Die Ausführungszeit behindert nicht meinen Entwicklerworkflow

Wie erkenne ich einen guten Test?

Ein guter Test gibt **schnelles**, **wertvolles** und **kosteneffizientes** Feedback zum Verhalten des Programms.

Schnell:

- Die Ausführungszeit behindert nicht meinen Entwicklerworkflow

Wertvoll:

- Es gibt keine false positives / negatives
- Es ist leicht nachvollziehbar, welches Szenario getestet wird
- Wenn der Test fehlschlägt ist leicht erkennbar, wie das Problem angegangen werden kann.

Wie erkenne ich einen guten Test?

Ein guter Test gibt **schnelles**, **wertvolles** und **kosteneffizientes** Feedback zum Verhalten des Programms.

Schnell:

- Die Ausführungszeit behindert nicht meinen Entwicklerworkflow

Wertvoll:

- Es gibt keine false positives / negatives
- Es ist leicht nachvollziehbar, welches Szenario getestet wird
- Wenn der Test fehlschlägt ist leicht erkennbar, wie das Problem angegangen werden kann.

Kosteneffizient:

- Der Aufwand den Test zu schreiben ist verhältnismäßig zur Entwicklung
- Der Aufwand den Test zu warten ist angemessen
- Die benötigte Infrastruktur ist angemessen

Arten von Tests…

- Funktionales Verhalten
  - Unit Test, Integrationstest, Akzeptanztest, UI-Test, Screenshot Test, End-to-End Test, Contract Test…
- Interne Qualität
  - Code Smells, Linting, Test Coverage, Mutation Coverage…
- Performance
- Security
  - Vulnerability Scan, Pen-Test…

Arten von Tests - Architektur

- Automatisierte Tests gibt es in vielen Arten.

Arten von Tests - Architektur

- Automatisierte Tests gibt es in vielen Arten.
- Für jede mögliche Art gibt es viele mögliche Technologien & Frameworks & 3rd-party-libraries.

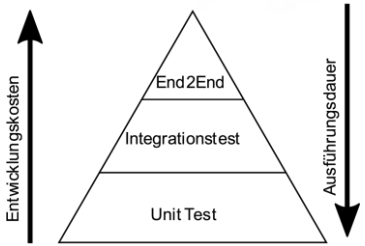
Arten von Tests - Architektur

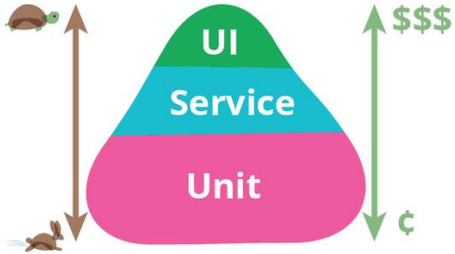
- Automatisierte Tests gibt es in vielen Arten.
- Für jede mögliche Art gibt es viele mögliche Technologien & Frameworks & 3rd-party-libraries.

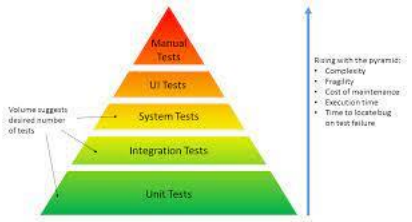
=> Wir müssen Entscheidungen treffen.

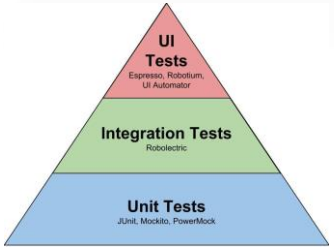
- Diese Entscheidungen sind nicht in Stein gemeißelt, aber
- Oft kostenaufwändig zu ändern / zu korrigieren
- Werden berücksichtigt wenn weitere Entscheidungen anstehen

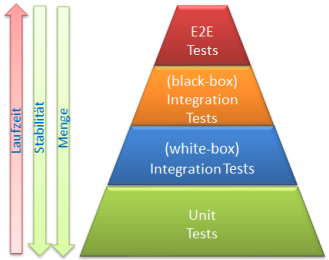
Die ideale Testarchitektur

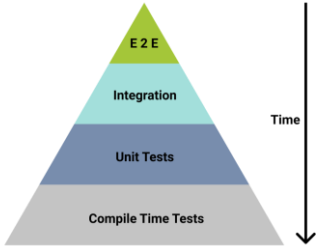


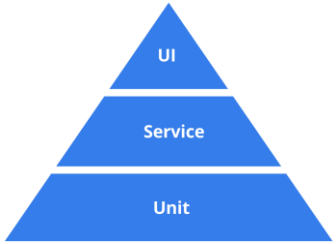


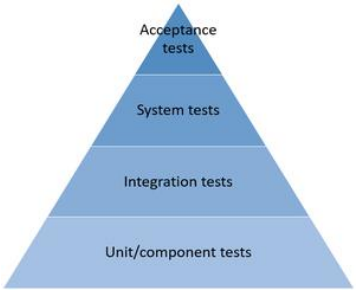




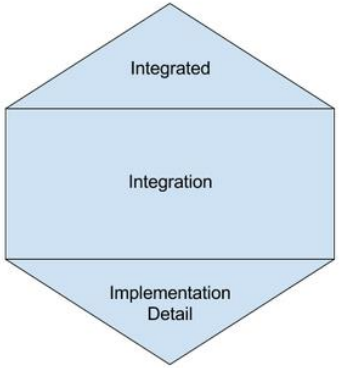




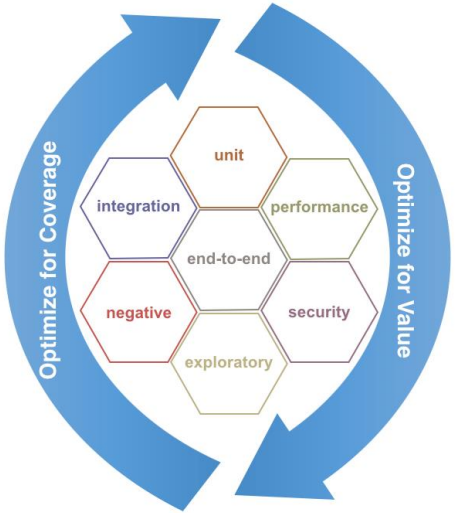




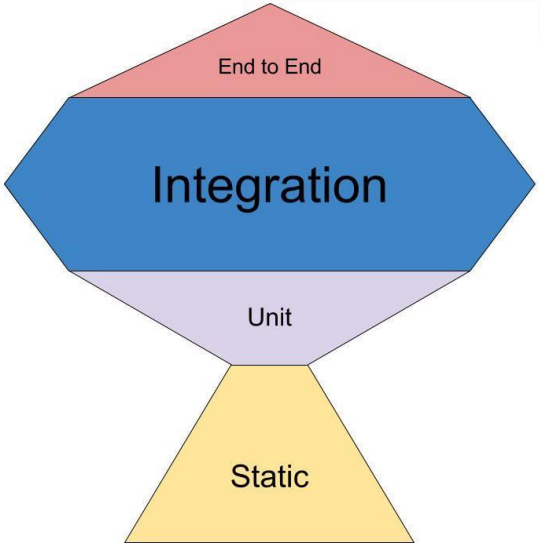
Die ideale Testarchitektur



Testing Honeycomb for Microservices - Spotify



Testing Honeycomb



Test Trophy

Die ideale Testarchitektur

Wenn ich etwas baue, dass wie ein Flugzeug aussieht, dann wird es nicht automatisch fliegen können.

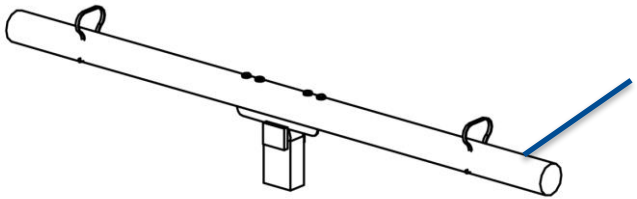
Wenn ich etwas baue, dass fliegen kann, dann *könnte* es wie ein Flugzeug aussehen.

Die ideale Testarchitektur

Wenn ich eine Suite baue, die wie eine Testpyramide aussieht, dann wird sie nicht automatisch schnelles, wertvolles und kosteneffizientes Feedback geben.

Wenn ich etwas baue, dass schnelles, wertvolles und kosteneffizientes Feedback gibt, dann *könnte* es wie eine Testpyramide aussehen.

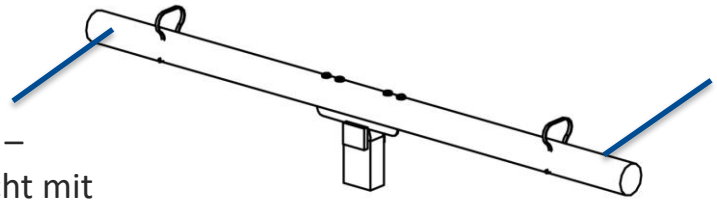
Emergente Testarchitektur – Die Balance



Architekturplan Cargo Cult – Kaum Inspect & Adapt Möglichkeiten ohne viel Diskussion und Konflikt. Es gibt keine Experimente. Der Plan wird eingehalten, die Tests könnten aber schneller, wertvoller und kosteneffizienter sein.

Emergente Testarchitektur – Die Balance

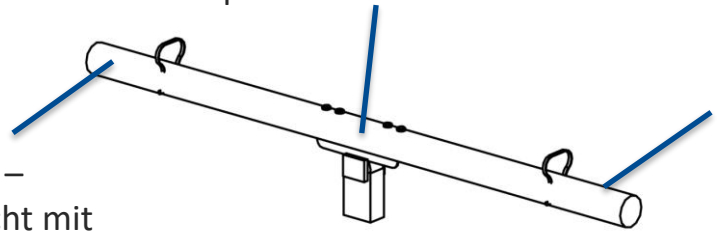
Accidental Architecture –
Man beschäftigt sich nicht mit dem Thema Testarchitektur. Leicht sichtbare Probleme werden über die Jahre immer schlimmer. Es fehlt die Konsequenz oder das Know-How für Inspect & Adapt.



Architekturplan Cargo Cult –
Kaum Inspect & Adapt Möglichkeiten ohne viel Diskussion und Konflikt. Es gibt keine Experimente. Der Plan wird eingehalten, die Tests könnten aber schneller, wertvoller und kosteneffizienter sein.

Emergente Testarchitektur – Die Balance

Emergente Testarchitektur – Vorhandene Vision mit Inspect & Adapt. Die Möglichkeiten und auch die gängigen Architekturen sind bekannt. Experimente finden statt.



Accidental Architecture – Man beschäftigt sich nicht mit dem Thema Testarchitektur. Leicht sichtbare Probleme werden über die Jahre immer schlimmer. Es fehlt die Konsequenz oder das Know-How für Inspect & Adapt.

Architekturplan Cargo Cult – Kaum Inspect & Adapt Möglichkeiten ohne viel Diskussion und Konflikt. Es gibt keine Experimente. Der Plan wird eingehalten, die Tests könnten aber schneller, wertvoller und kosteneffizienter sein.

Emergente Testarchitektur - Inspect

- **Schnell**
  -
  -
- **Wertvoll**
  -
  -
  -
  -
  -
- **Kosteneffizient**
  -
  -
  -
  -

Emergente Testarchitektur - Inspect

- **Schnell**
  - **Repeatability** – Wie oft führen wir diese Suite aus?
  - **Speed** – Wie lange warten wir auf Feedback?
- **Wertvoll**
  -
  -
  -
  -
  -
- **Kosteneffizient**
  -
  -
  -
  -

Emergente Testarchitektur - Inspect

- **Schnell**
  - **Repeatability** – Wie oft führen wir diese Suite aus?
  - **Speed** – Wie lange warten wir auf Feedback?
- **Wertvoll**
  - **Stability** – Laufen die Tests zuverlässig?
  - **Coverage** – Was decken wir mit dieser Suite ab?
  - **Useful Failures** – Schlagen die Tests aus hilfreichen Gründen fehl
  - **Debugability** – Wie leicht fixen wir Fehler wenn ein Test failt?
  - **Readability** – Wie leicht kann das Verhalten des Programms im Test nachvollzogen werden?
- **Kosteneffizient**
  -
  -
  -
  -

Emergente Testarchitektur - Inspect

- **Schnell**
  - **Repeatability** – Wie oft führen wir diese Suite aus?
  - **Speed** – Wie lange warten wir auf Feedback?
- **Wertvoll**
  - **Stability** – Laufen die Tests zuverlässig?
  - **Coverage** – Was decken wir mit dieser Suite ab?
  - **Useful Failures** – Schlagen die Tests aus hilfreichen Gründen fehl?
  - **Debugability** – Wie leicht fixen wir Fehler wenn ein Test failt?
  - **Readability** – Wie leicht kann das Verhalten des Programms im Test nachvollzogen werden?
- **Kosteneffizient**
  - **Development** – Wieviel Implementieraufwand wird uns diese Testschicht kosten?
  - **Cognitive Load** – Wieviel komplizierter wird das Produkt mit dieser Testschicht?
  - **Infrastructure** – Welche Infrastruktur wird benötigt & wie teuer ist die?
  - **Maintenance** – Wie oft & wie teuer müssen wir die Testsuite warten?

Emergente Testarchitektur - Inspect

MyItems - Testschichten	Mehrwert	Kosten	Debugability	Cognitive Load	Stability	Useful Failure
Unit Tests						
Integration Test						
Contract Tests						
End-to-End Tests						
Performance Tests						

Emergente Testarchitektur

Wieso?

Emergente Testarchitektur

Wieso?

- Schnelles, wertvolles & kosteneffizientes Feedback

Emergente Testarchitektur

Wieso?

- Schnelles, wertvolles & kosteneffizientes Feedback

Welche Schichten?

Emergente Testarchitektur

Wieso?

- Schnelles, wertvolles & kosteneffizientes Feedback

Welche Schichten?

- It depends

Emergente Testarchitektur

Wieso?

- Schnelles, wertvolles & kosteneffizientes Feedback

Welche Schichten?

- It depends

Wie?

Emergente Testarchitektur

Wieso?

- Schnelles, wertvolles & kosteneffizientes Feedback

Welche Schichten?

- It depends

Wie?

- Bewusst Zeit für Inspect nehmen. Visualisierung hilft.

Emergente Testarchitektur

Wieso?

- Schnelles, wertvolles & kosteneffizientes Feedback

Welche Schichten?

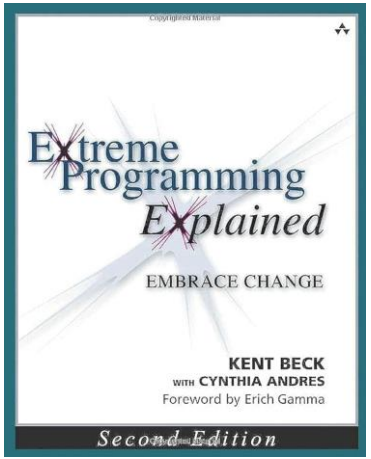
- It depends

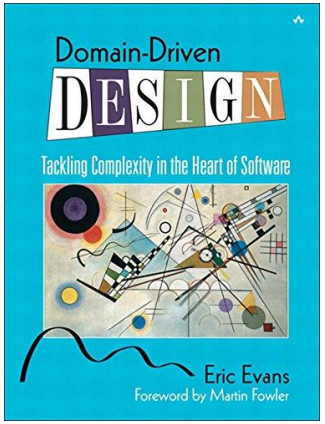
Wie?

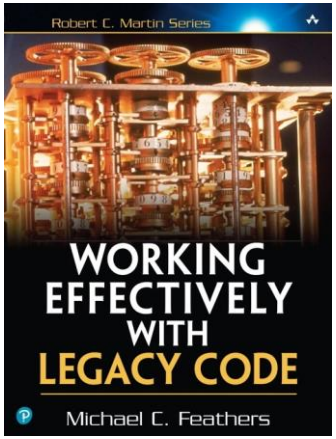
- Bewusst Zeit für Inspect nehmen. Visualisierung hilft.
- 

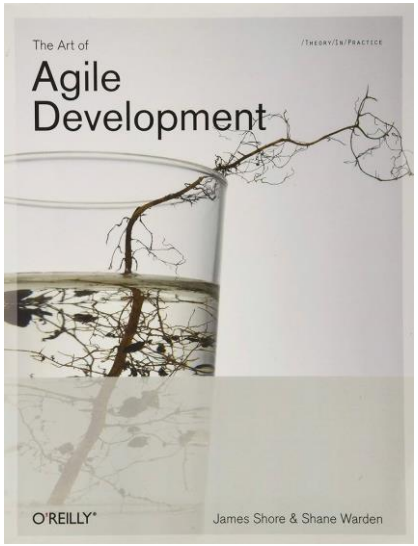
Emergente Testarchitektur

Wie?









Die Architektur meiner Testsuite

Die Basics, die wir immer wieder vergessen.

Moritz Tiedje

moritz.tiedje@andrena.de

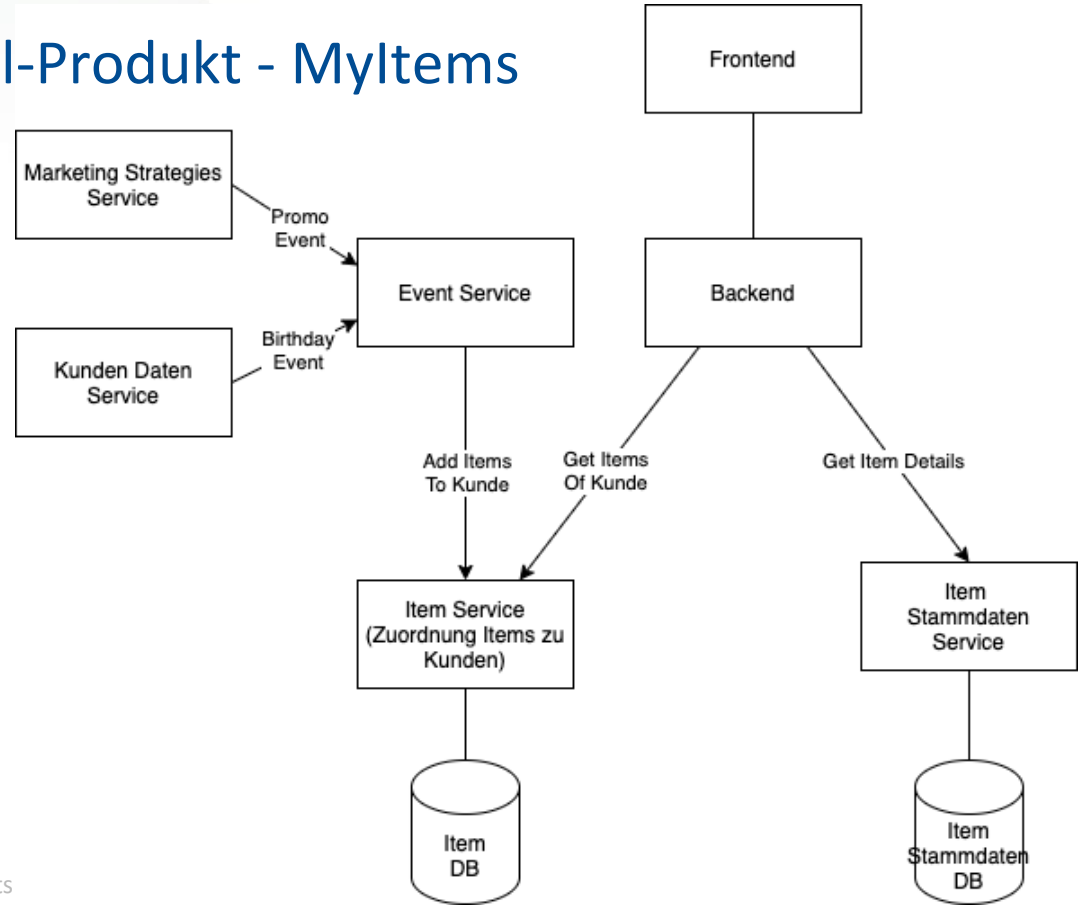
Warum schreiben wir Tests?

- Freiheit von hohem manuellen Testaufwand
- Lebendige Dokumentation
- Leichte Wartbarkeit
- Anforderungsdefinition (TDD)
- Grundbaustein im Entwicklerworkflow:
  - Write Failing Test -> Make it pass -> Refactor (TDD)
  - Get it working –> Test it –> Refactor

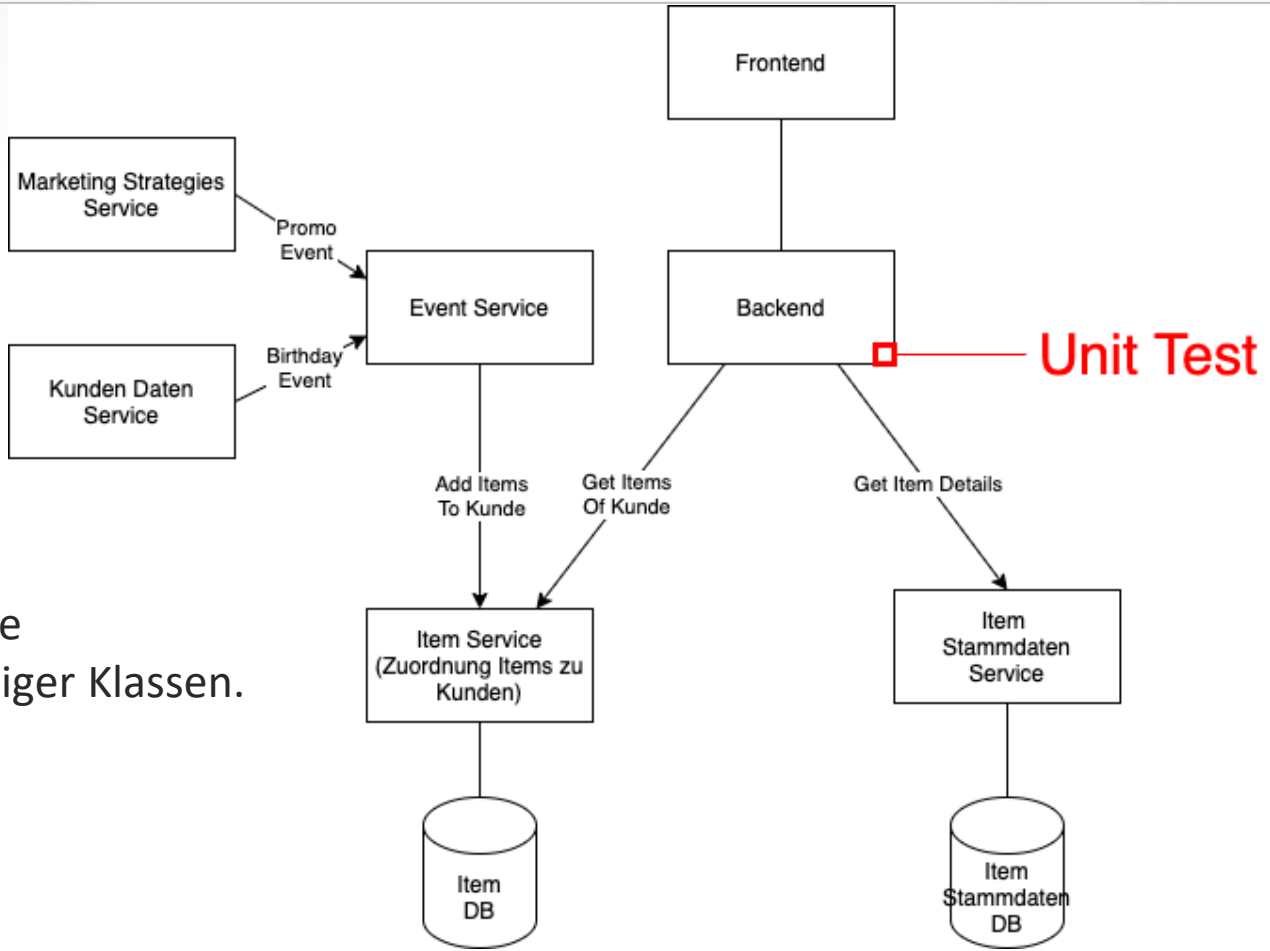
Agenda

1. Testtypen Vorstellungsrunde
2. Wie sieht die ideale Testsuite aus?
3. Wie bewerte ich eine Testsuite?
4. Ein paar Projekt - Beispiele

Beispiel-Produkt - MyItems

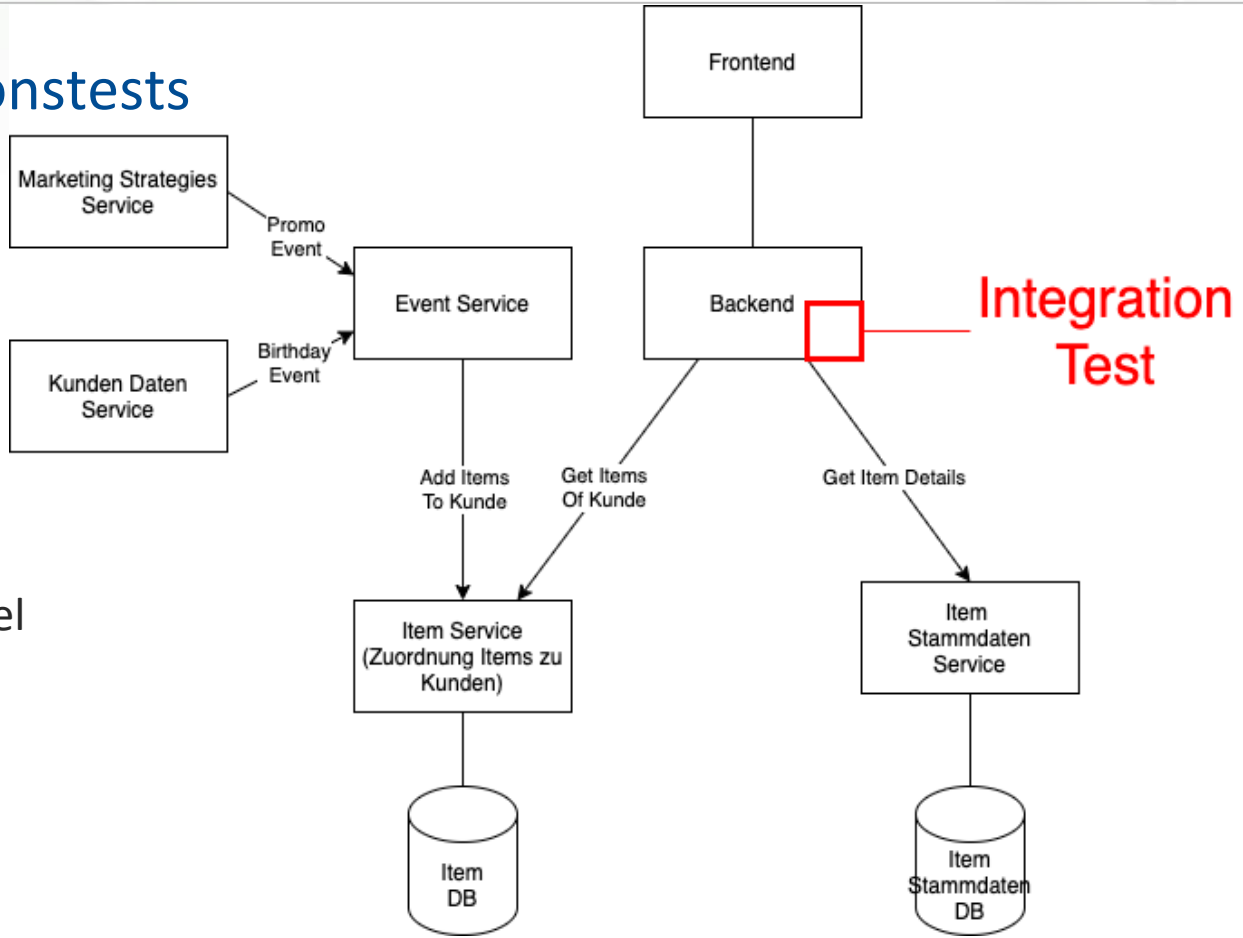


Unit Tests



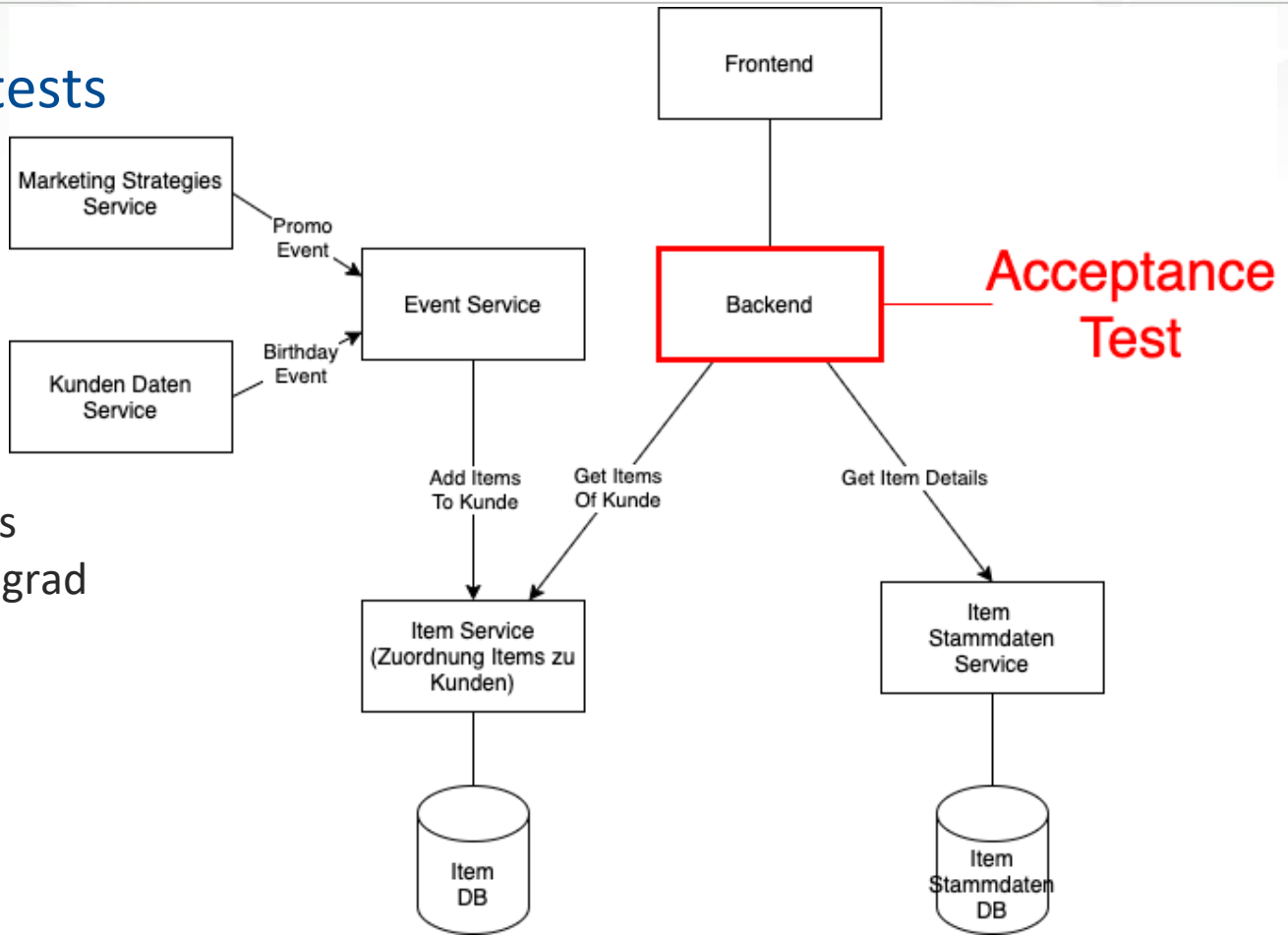
Funktionsweise einzelner/weniger Klassen.

Integrationstests



Zusammenspiel von Units

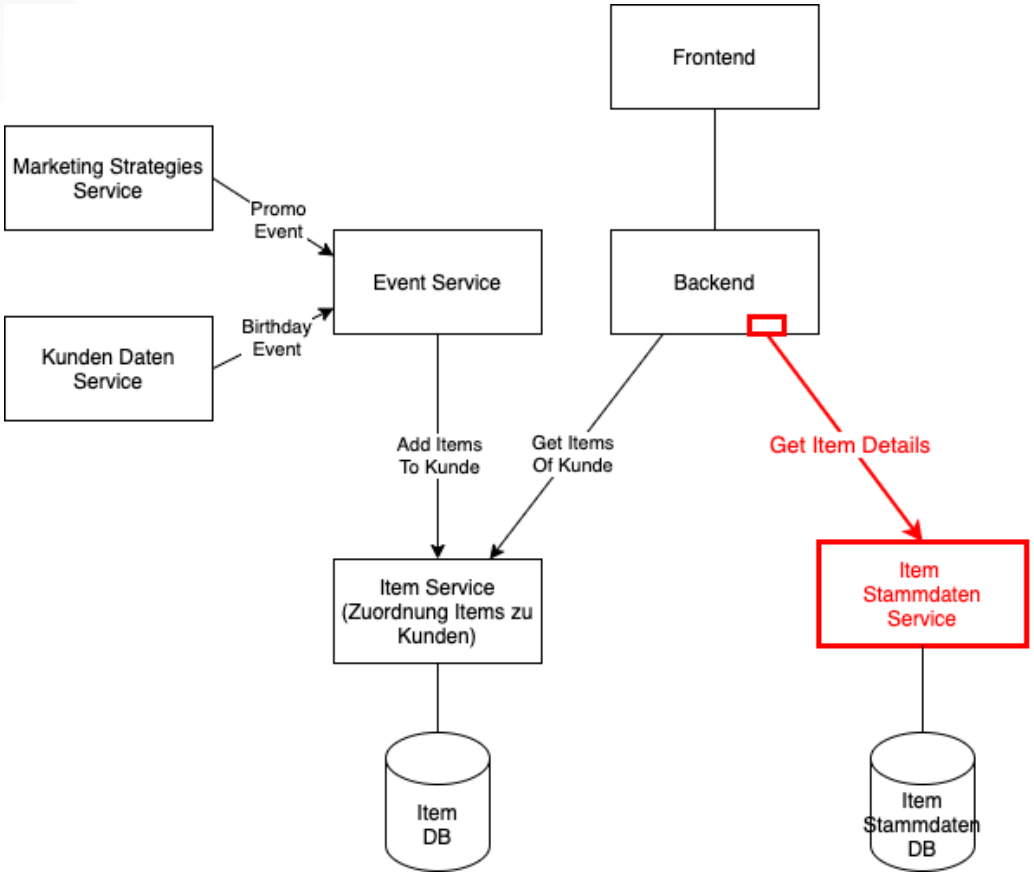
Akzeptanztests



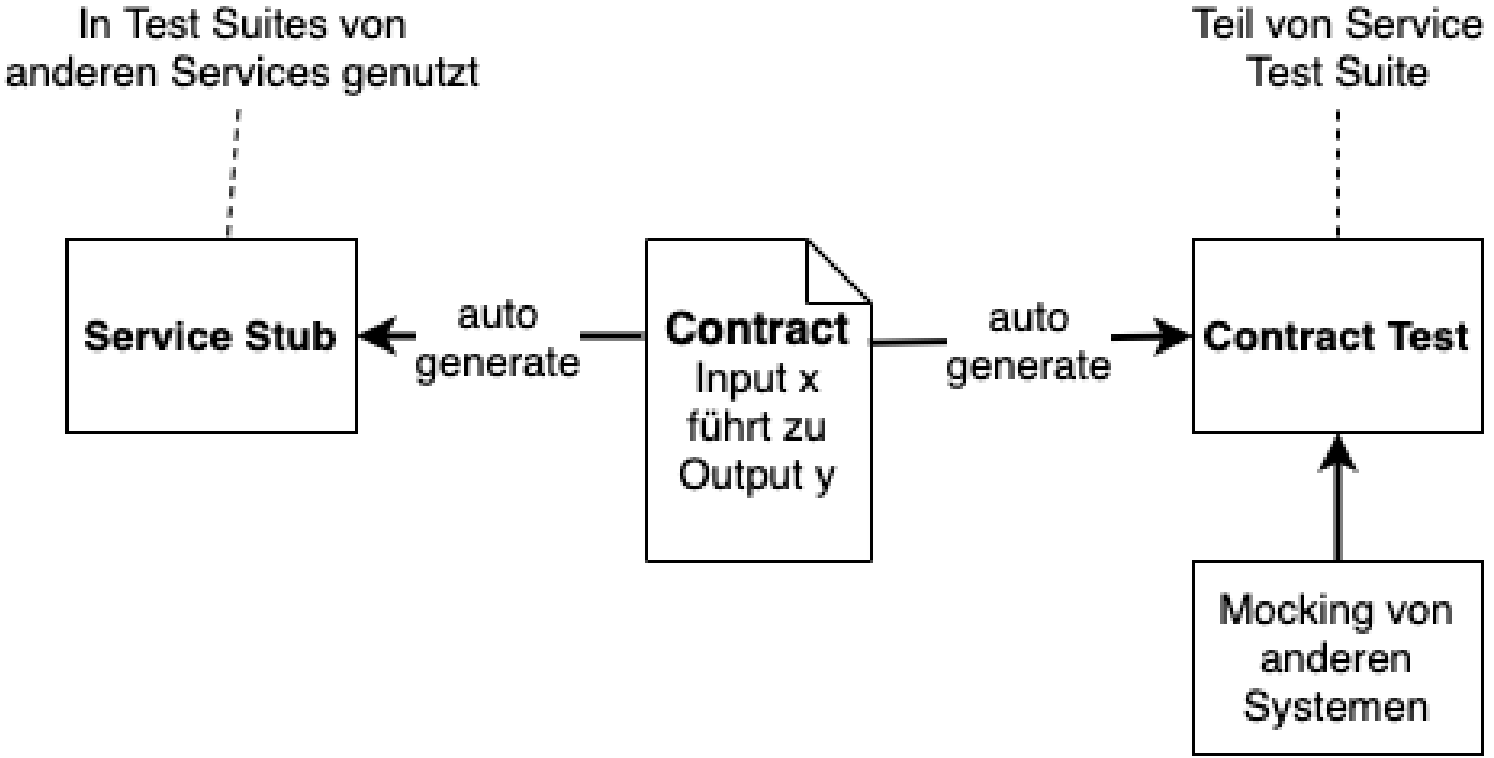
Fachlicher Fokus
Niedriger Detailgrad
Gesamte App

z.B. Cucumber

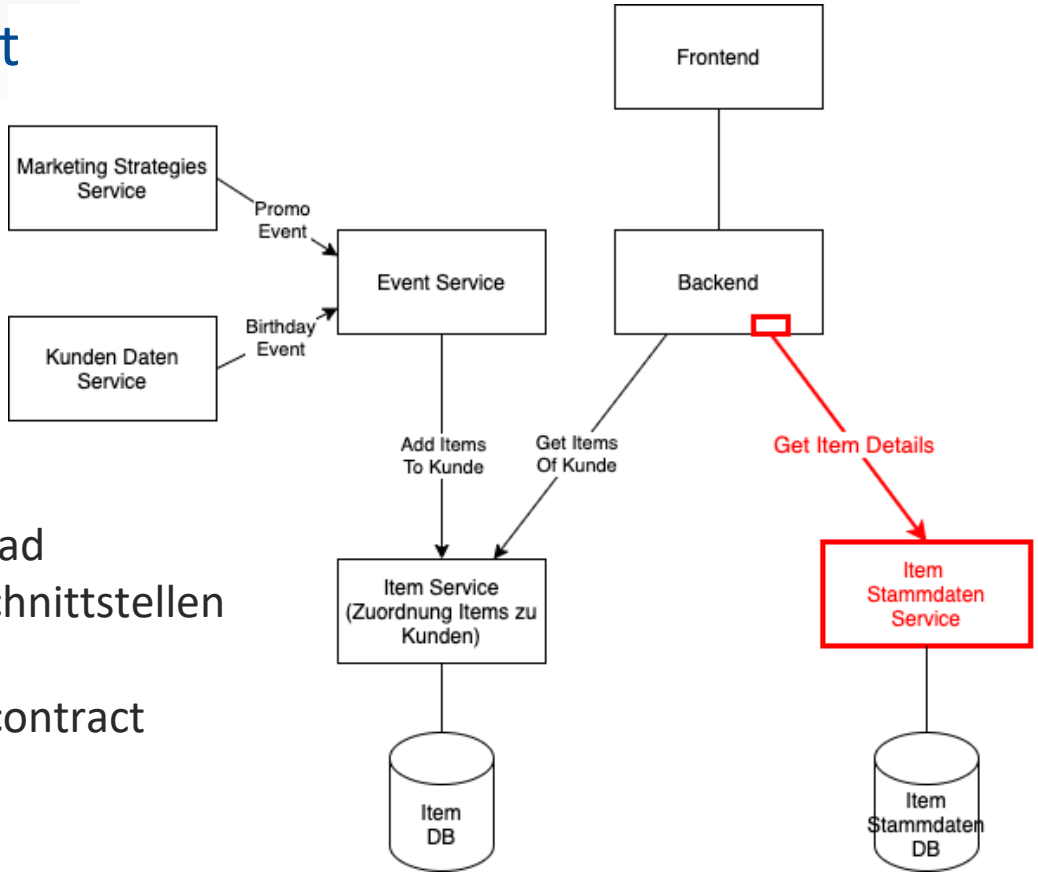
Contract



Contract Test – Wie es funktioniert



Contract Test

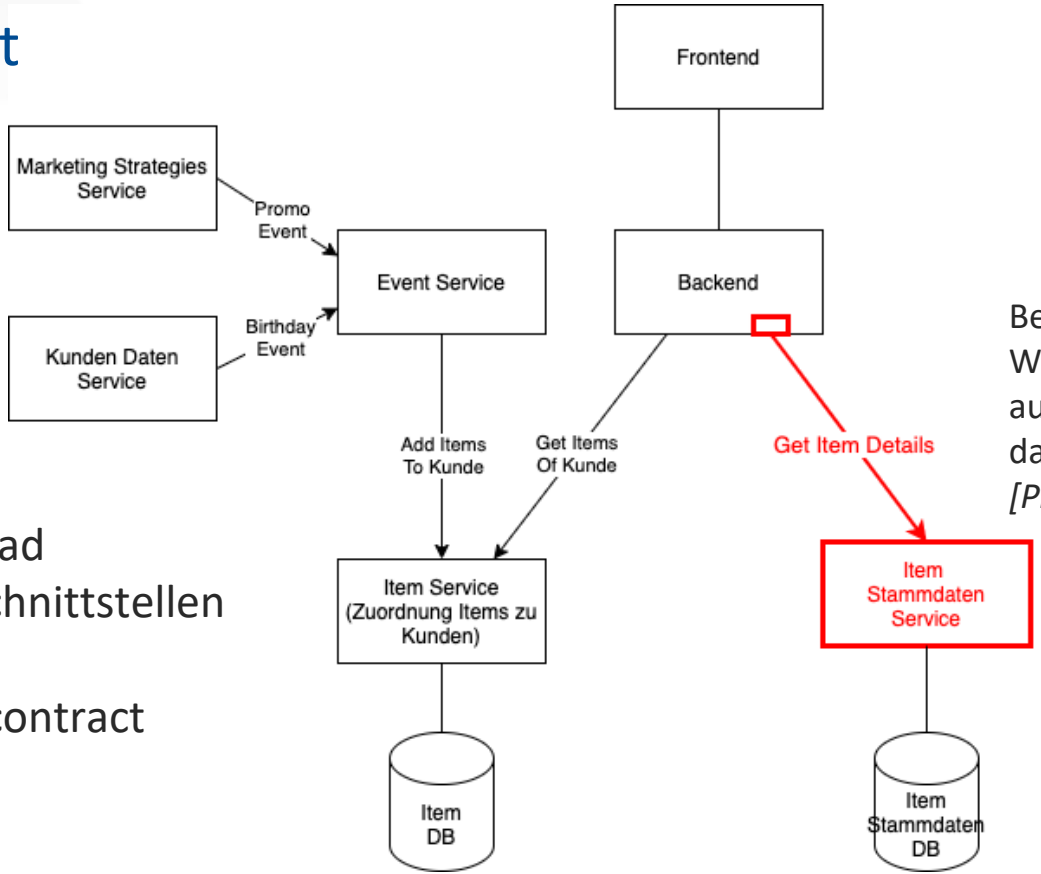


Contract Test

Fachlicher Fokus
Niedriger Detailgrad
Gesamte App + Schnittstellen

z.B. Spring cloud contract

Contract Test



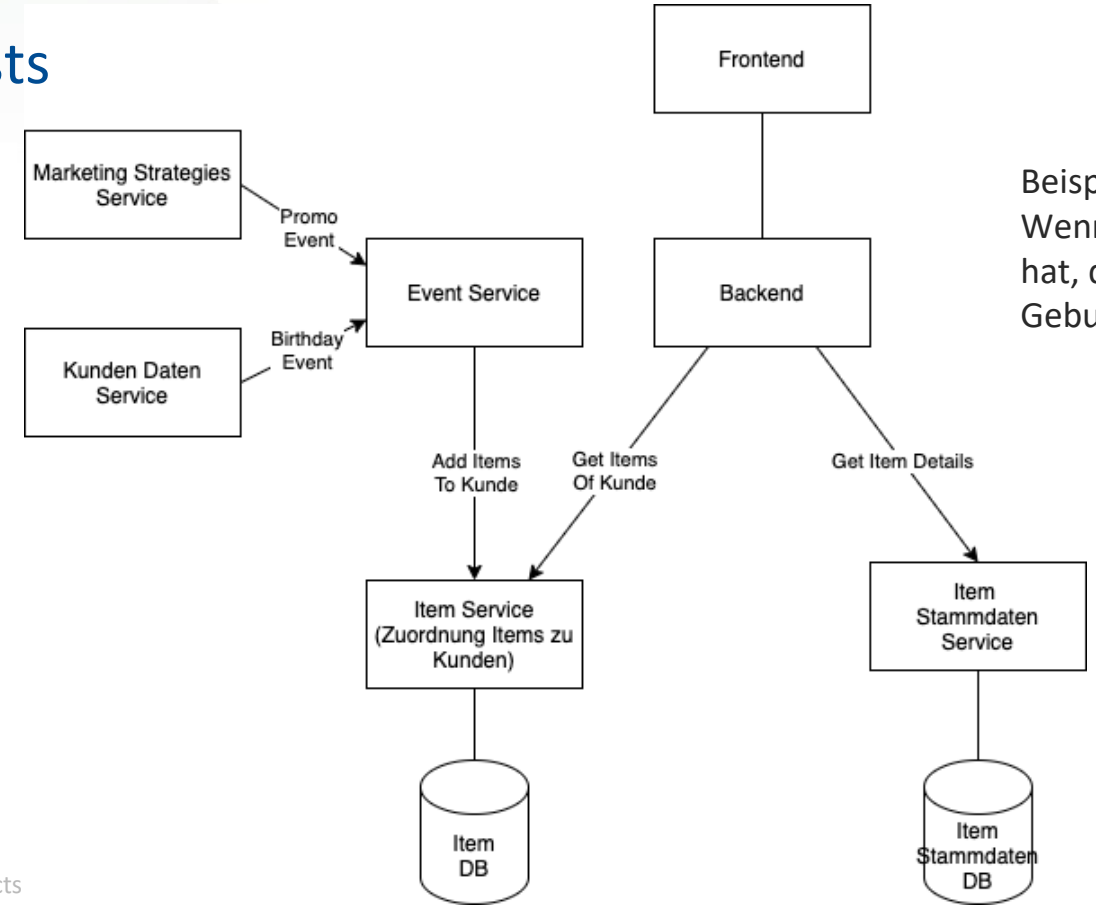
Contract Test

Beispiel Contract:
Wenn ich *Get Item Details* aufrufe mit dem Item *Teddy* dann lautet die Antwort *[Preis: 5 Euro, Typ: Spielzeug]*

Fachlicher Fokus
Niedriger Detailgrad
Gesamte App + Schnittstellen

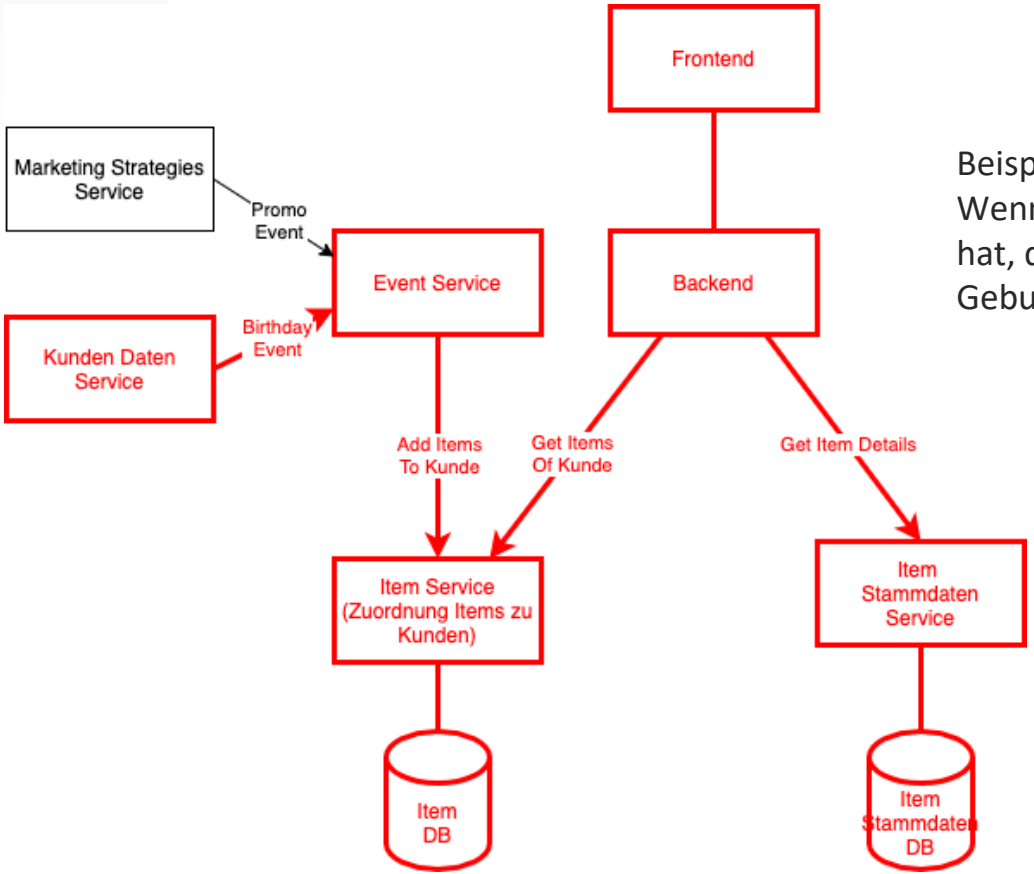
z.B. Spring cloud contract

Livetests



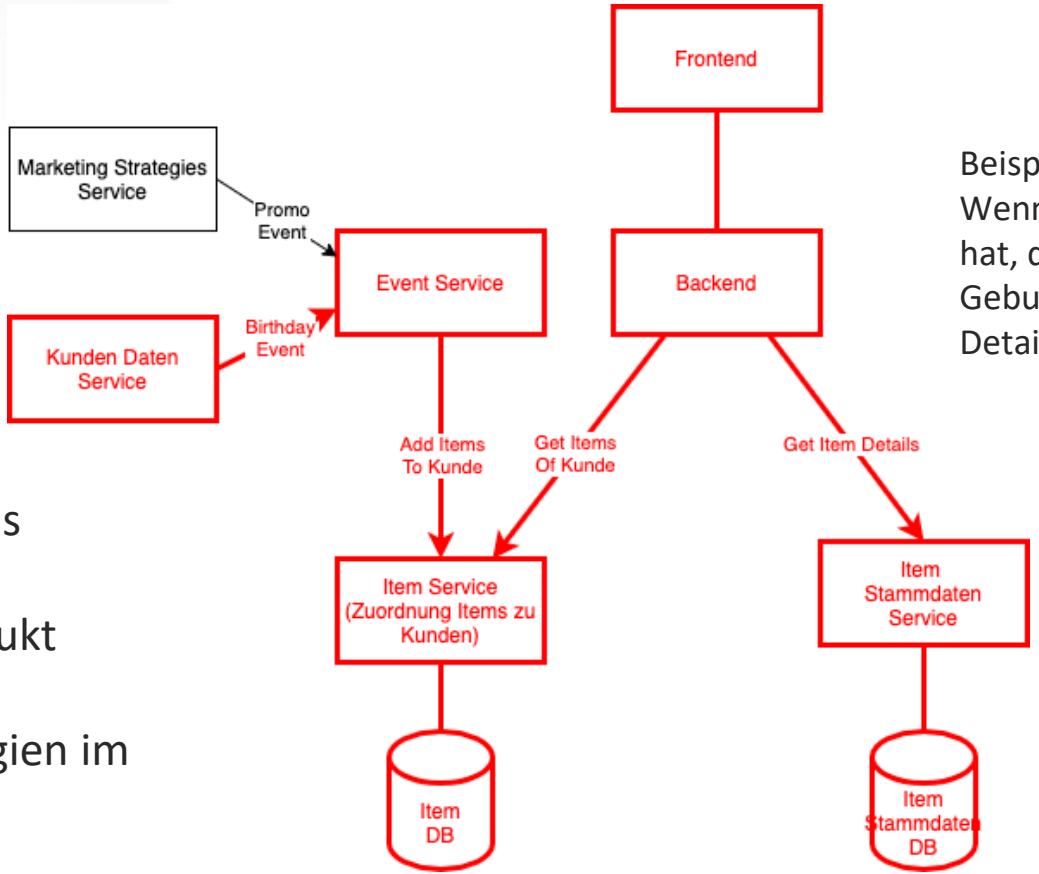
Beispiel Live Test:
Wenn ein Kunde Geburtstag hat, dann bekommt er das Geburtstagshut Item.

Livetests



Beispiel Live Test:
Wenn ein Kunde Geburtstag hat, dann bekommt er das Geburtstagshut Item.

Livetests

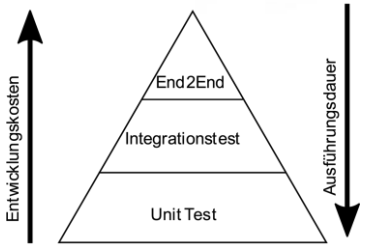


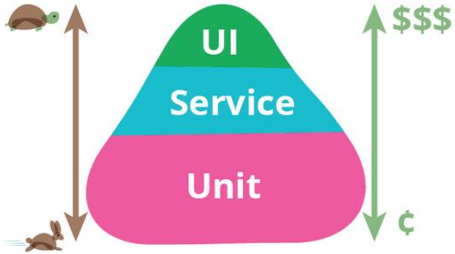
Beispiel Live Test:
Wenn ein Kunde Geburtstag hat, dann kann er das Geburtstagshut Item mit allen Details im Frontend sehen.

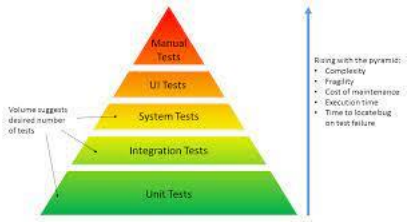
Fachlicher Fokus
Keine Details
Gesamtes Produkt

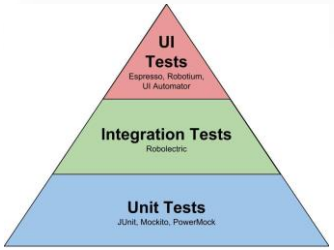
Viele Technologien im Einsatz

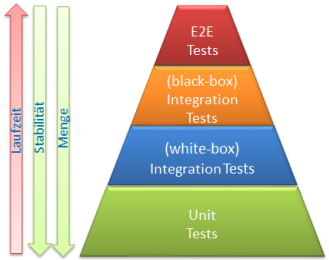
Die ideale Testsuite - Ein Blick in die Literatur

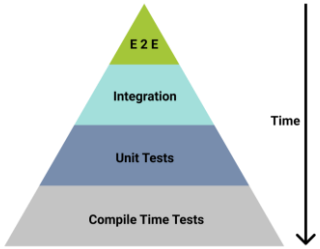


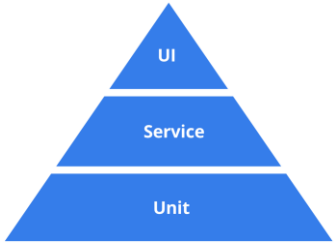


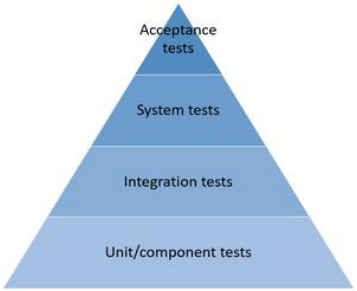




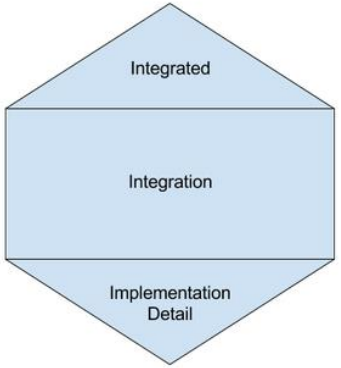


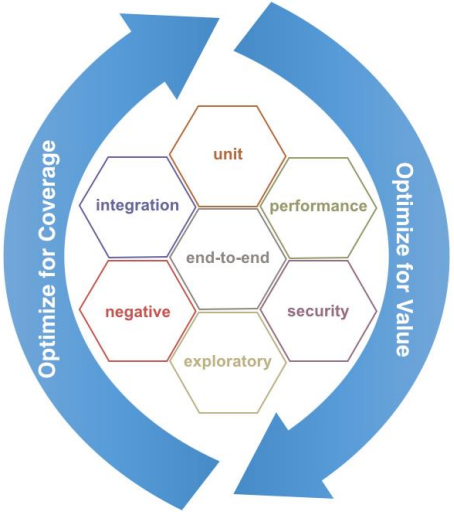


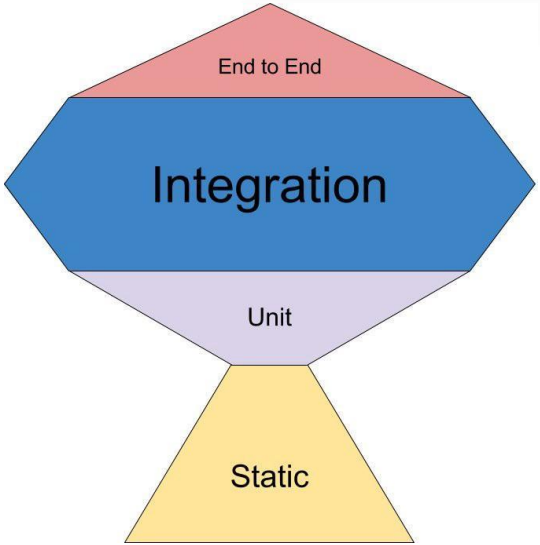




Die ideale Testsuite - Ein Blick in die Literatur





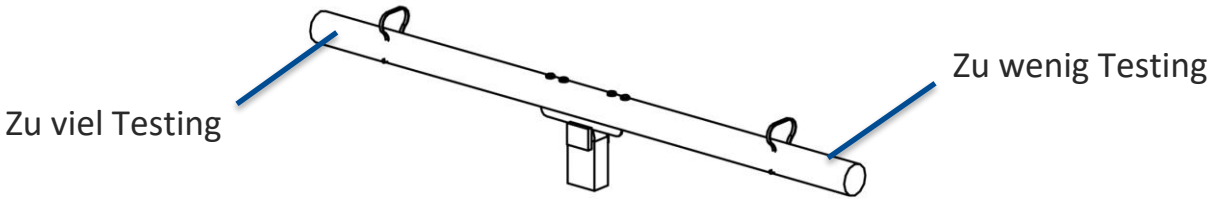


Also, wie soll die ideale Testsuite aussehen?

Jede vorgestellte Testart hat eigenen Scope, einzigartige Mehrwerte, unterschiedliche Kosten

Also, wie soll die ideale Testsuite aussehen?

Jede vorgestellte Testart hat eigenen Scope, einzigartige Mehrwerte, unterschiedliche Kosten

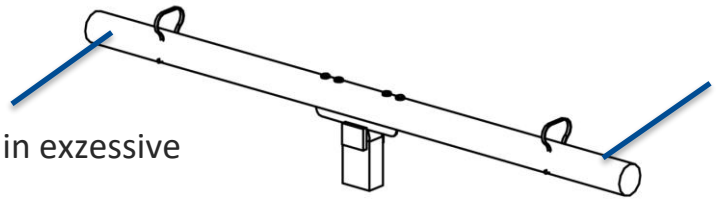


Also, wie soll die ideale Testsuite aussehen?

Jede vorgestellte Testart hat eigenen Scope, einzigartige Mehrwerte, unterschiedliche Kosten

Zu viel Testing:

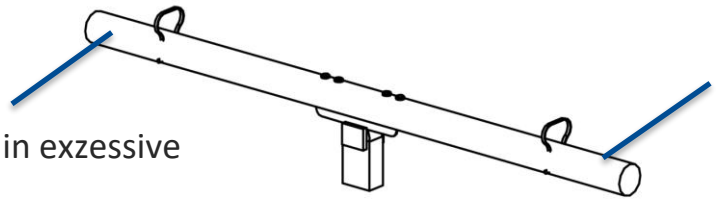
- Geldverschwendung in exzessive Qualität



Zu wenig Testing

Also, wie soll die ideale Testsuite aussehen?

Jede vorgestellte Testart hat eigenen Scope, einzigartige Mehrwerte, unterschiedliche Kosten



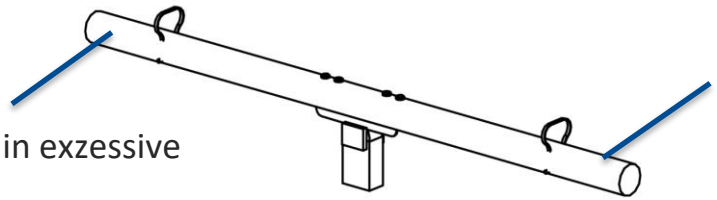
Zu wenig Testing

Zu viel Testing:

- Geldverschwendung in exzessive Qualität
- Hoher Cognitive Load von dem Projekt, wenn zu viele Test-Frameworks bekannt sein müssen.

Also, wie soll die ideale Testsuite aussehen?

Jede vorgestellte Testart hat eigenen Scope, einzigartige Mehrwerte, unterschiedliche Kosten



Zu wenig Testing

Zu viel Testing:

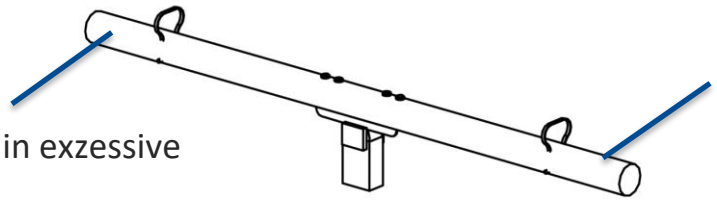
- Geldverschwendung in exzessive Qualität
- Hoher Cognitive Load von dem Projekt, wenn zu viele Test-Frameworks bekannt sein müssen.
- „Test Fetischismus“

Also, wie soll die ideale Testsuite aussehen?

Jede vorgestellte Testart hat eigenen Scope, einzigartige Mehrwerte, unterschiedliche Kosten

Zu wenig Testing:

- Bugs

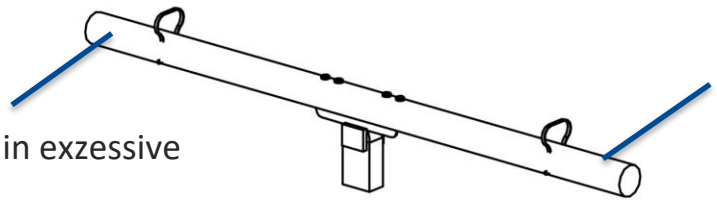


Zu viel Testing:

- Geldverschwendung in exzessive Qualität
- Hoher Cognitive Load von dem Projekt, wenn zu viele Test-Frameworks bekannt sein müssen.
- „Test Fetischismus“

Also, wie soll die ideale Testsuite aussehen?

Jede vorgestellte Testart hat eigenen Scope, einzigartige Mehrwerte, unterschiedliche Kosten



Zu viel Testing:

- Geldverschwendung in exzessive Qualität
- Hoher Cognitive Load von dem Projekt, wenn zu viele Test-Frameworks bekannt sein müssen.
- „Test Fetischismus“

Zu wenig Testing:

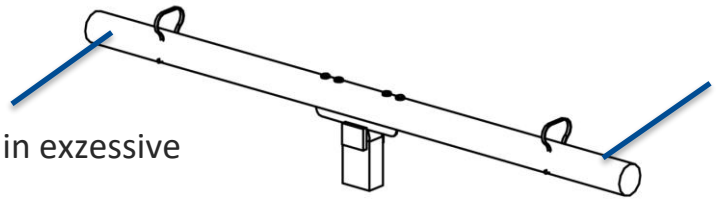
- Bugs
- Legacy-Code

Also, wie soll die ideale Testsuite aussehen?

Jede vorgestellte Testart hat eigenen Scope, einzigartige Mehrwerte, unterschiedliche Kosten

Zu viel Testing:

- Geldverschwendung in exzessive Qualität
- Hoher Cognitive Load von dem Projekt, wenn zu viele Test-Frameworks bekannt sein müssen.
- „Test Fetischismus“



Zu wenig Testing:

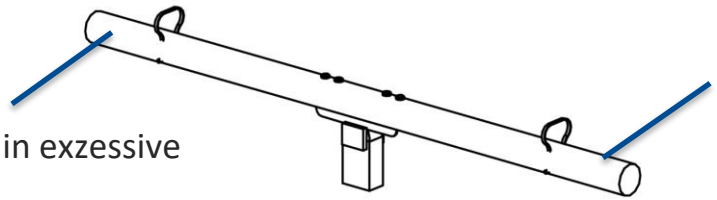
- Bugs
- Legacy-Code
- Langfristig wird Entwicklung deutlich teurer

Also, wie soll die ideale Testsuite aussehen?

Jede vorgestellte Testart hat eigenen Scope, einzigartige Mehrwerte, unterschiedliche Kosten

Zu viel Testing:

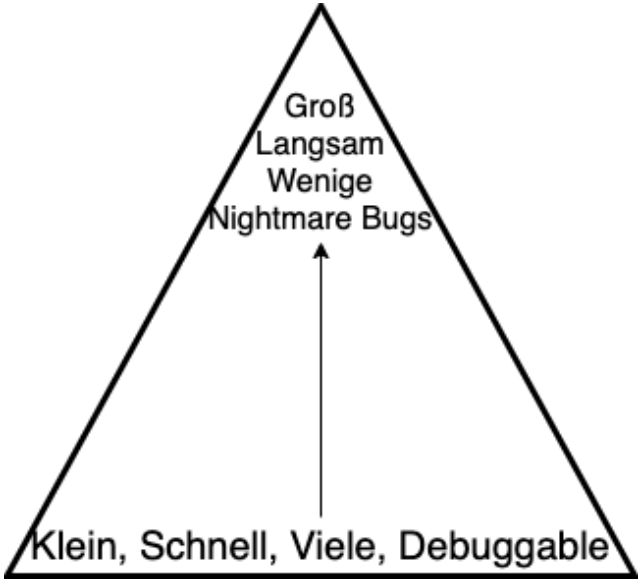
- Geldverschwendung in exzessive Qualität
- Hoher Cognitive Load von dem Projekt, wenn zu viele Test-Frameworks bekannt sein müssen.
- „Test Fetischismus“



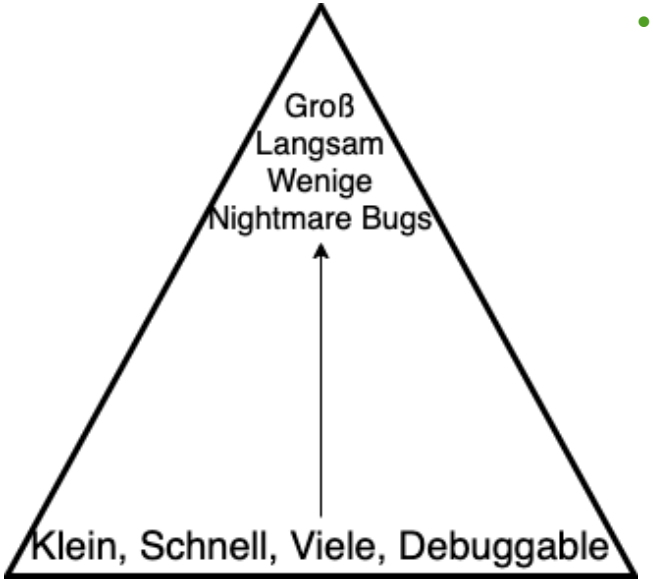
Zu wenig Testing:

- Bugs
- Legacy-Code
- Langfristig wird Entwicklung deutlich teurer
- „Passt schon, läuft doch“

Die Ideale Test Suite – „It depends“

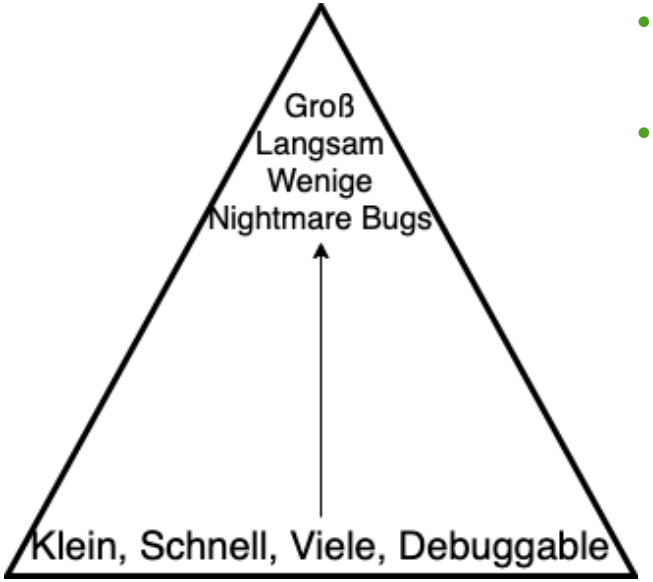


Die Ideale Test Suite – „It depends“



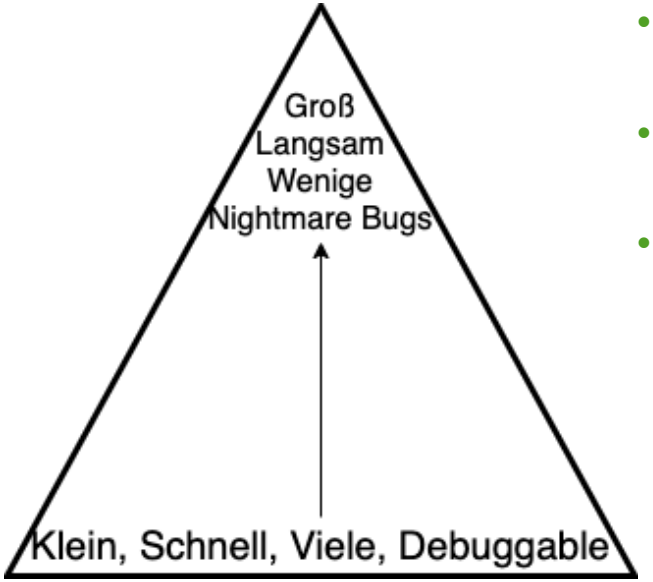
- Grundsätzlich – Die Pyramide hat sich bisher in der Testarchitektur bewährt.

Die Ideale Test Suite – „It depends“



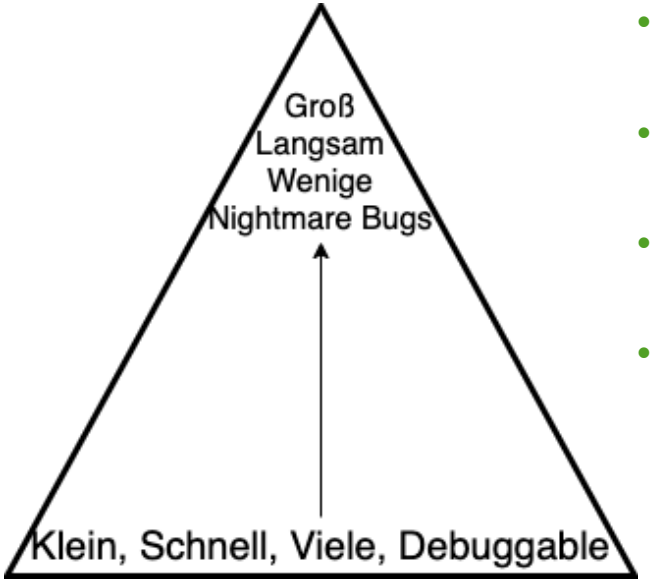
- Grundsätzlich – Die Pyramide hat sich bisher in der Testarchitektur bewährt.
- Es gibt zahlreiche Formen von Testschichten in der Literatur, die hier nicht vorkamen. (Compile-Time, A/B, …)

Die Ideale Test Suite – „It depends“



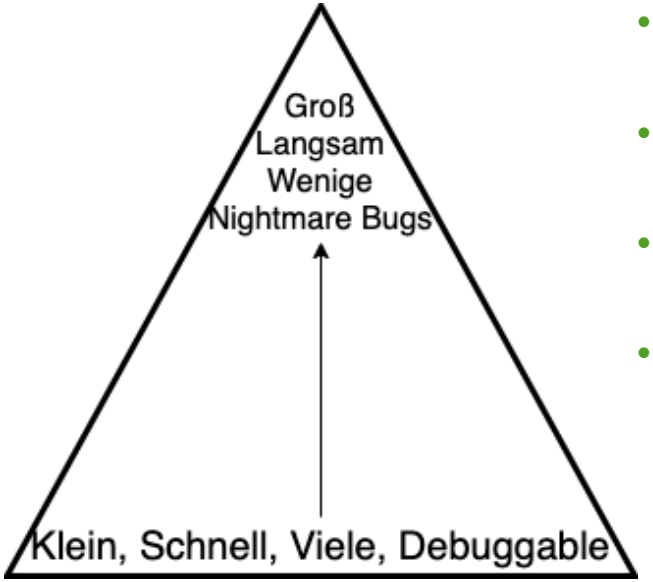
- Grundsätzlich – Die Pyramide hat sich bisher in der Testarchitektur bewährt.
- Es gibt zahlreiche Formen von Testschichten in der Literatur, die hier nicht vorkamen. (Compile-Time, A/B, …)
- Nicht jede Testschicht ist auf jedes Produkt anwendbar oder dafür geeignet.

Die Ideale Test Suite – „It depends“



- Grundsätzlich – Die Pyramide hat sich bisher in der Testarchitektur bewährt.
- Es gibt zahlreiche Formen von Testschichten in der Literatur, die hier nicht vorkamen. (Compile-Time, A/B, …)
- Nicht jede Testschicht ist auf jedes Produkt anwendbar oder dafür geeignet.
- Viel Produkte entwickeln über die Zeit eine mehr oder weniger gute „Accidental Test Architecture“

Die Ideale Test Suite – „It depends“



- Grundsätzlich – Die Pyramide hat sich bisher in der Testarchitektur bewährt.
- Es gibt zahlreiche Formen von Testschichten in der Literatur, die hier nicht vorkamen. (Compile-Time, A/B, …)
- Nicht jede Testschicht ist auf jedes Produkt anwendbar oder dafür geeignet.
- Viel Produkte entwickeln über die Zeit eine mehr oder weniger gute „Accidental Test Architecture“

Welche Schichten sind den Aufwand wert, welche nicht?

Testschichten – Woran man bei der Architektur denken sollte

- **Definition** – Was für Tests sollen in dieser Testschicht sein?

Testschichten – Woran man bei der Architektur denken sollte

- **Definition** – Was für Tests sollen in dieser Testschicht sein?
- **Mehrwert** – Welche Vorteile bringt uns diese Schicht?

Testschichten – Woran man bei der Architektur denken sollte

- **Definition** – Was für Tests sollen in dieser Testschicht sein?
- **Mehrwert** – Welche Vorteile bringt uns diese Schicht?
- **Kosten** – Wieviel Arbeit wird uns diese Testschicht kosten?

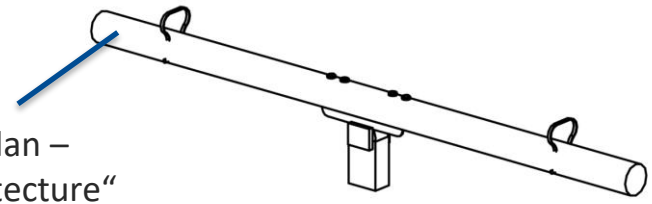
Testschichten – Woran man bei der Architektur denken sollte

- **Definition** – Was für Tests sollen in dieser Testschicht sein?
- **Mehrwert** – Welche Vorteile bringt uns diese Schicht?
- **Kosten** – Wieviel Arbeit wird uns diese Testschicht kosten?
- **Debugability** – Wie leicht finden wir Fehler mit dieser Testschicht?

Testschichten – Woran man bei der Architektur denken sollte

- **Definition** – Was für Tests sollen in dieser Testschicht sein?
- **Mehrwert** – Welche Vorteile bringt uns diese Schicht?
- **Kosten** – Wieviel Arbeit wird uns diese Testschicht kosten?
- **Debugability** – Wie leicht fixen wir Fehler wenn ein Test failt?
- **Cognitive Load** – Wieviel komplizierter wird das Produkt mit dieser Testschicht?
- **Stability** – Laufen die Tests zuverlässig durch?
- **Useful Failures** – Schlagen die Test aus hilfreichen Gründen fehl?

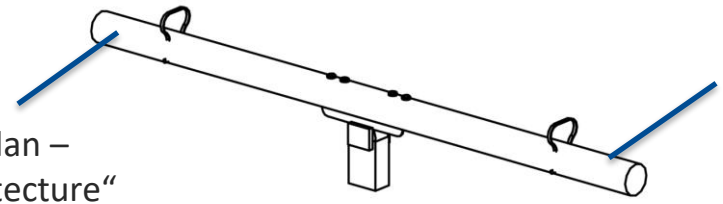
Emergente Testarchitektur



Kein Architekturplan –
„Accidental Architecture“

Emergente Testarchitektur

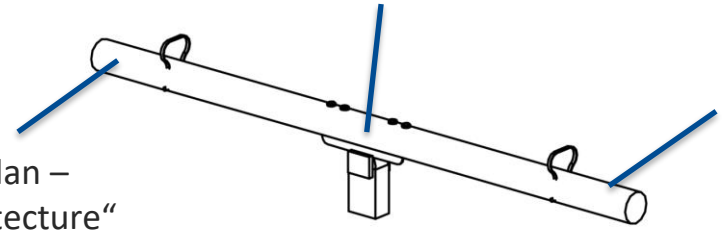
Kein Architekturplan –
„Accidental Architecture“



Zu viel Plan – Keine Inspect &
Adapt Möglichkeiten. (Wasserfall)

Emergente Testarchitektur

Emergente Architektur – Vorhandene Vision mit Inspect & Adapt

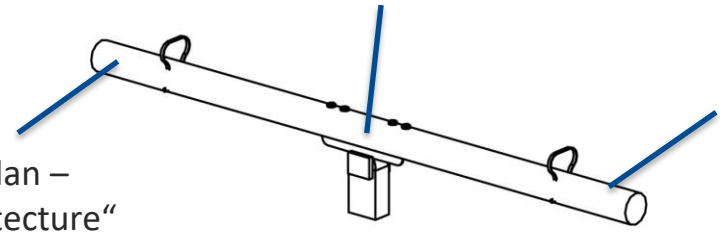


Zu viel Plan – Keine Inspect & Adapt Möglichkeiten. (Wasserfall)

Kein Architekturplan – „Accidental Architecture“

Emergente Testarchitektur

Emergente Architektur – Vorhandene Vision mit Inspect & Adapt

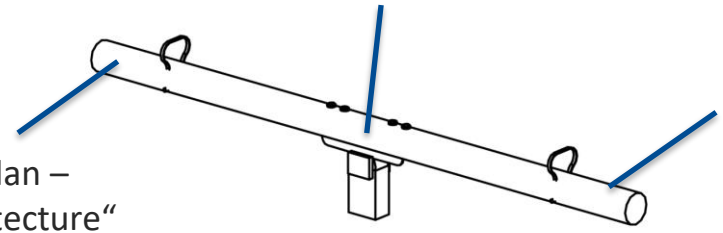


Zu viel Plan – Keine Inspect & Adapt Möglichkeiten. (Wasserfall)

Kein Architekturplan – „Accidental Architecture“

Emergente Testarchitektur

Emergente Architektur – Vorhandene Vision mit Inspect & Adapt

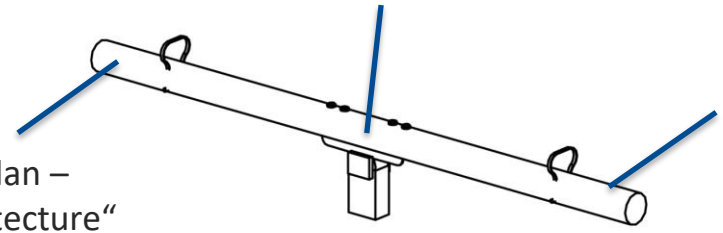


Zu viel Plan – Keine Inspect & Adapt Möglichkeiten. (Wasserfall)

Kein Architekturplan – „Accidental Architecture“

Emergente Testarchitektur

Emergente Architektur – Vorhandene Vision mit Inspect & Adapt



Zu viel Plan – Keine Inspect & Adapt Möglichkeiten. (Wasserfall)

Kein Architekturplan – „Accidental Architecture“

Bewertungskriterien einer Testsuite

Mehrwert:

Welchen Mehrwert bringt diese Suite gegenüber den anderen?

- Schnelle Laufzeit
- Aussage über Performance
- Integrative Testabdeckung
- Testet die UI / Performance / Architektur…

Kosten:

Emergente Testarchitektur - Inspect

MyItems - Testschichten	Mehrwert	Kosten	Debugability	Cognitive Load	Stability	Useful Failure
Unit Tests						
Integration Test						
Contract Tests						
End-to-End Tests						
Performance Tests						

Emergente Testarchitektur - Adapt

- Können wir die Pain Points einer Testschicht reduzieren?
- Wollen wir diese Schichten weiter warten?
- Wollen wir diese Schichten erweitern?
- Lohnt es sich nochmal in Infrastruktur zu investieren?
- Sind wir mit unserer Architektur auf dem richtigen Weg?
- Wollen wir ein Experiment machen?
- Wie erfolgreich waren unsere Experimente?
- …

Keine dieser Fragen sind unüblich in einem Inspect & Adapt Prozess

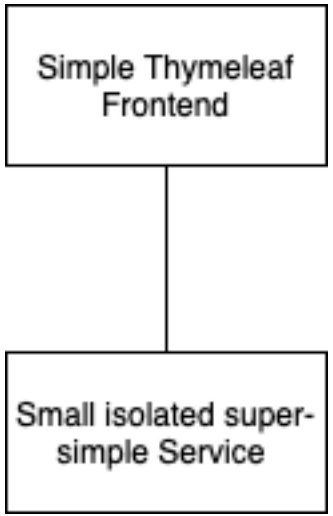
FAZIT

- Es gibt viele Arten von Testschichten, mit eigenen Vor- und Nachteilen.
- Es gibt viele Vorschläge aus der Literatur, wie eine Architektur aussehen könnte.
- Sowohl bei der Umsetzung als auch bei der Planung der Architektur ist ein gesundes Mittelmaß an Aufwand wichtig.
- Die ideale Testarchitektur ist emergent und stark projektabhängig.
- Testarchitektur muss einem konstanten Inspect & Adapt – Prozess unterliegen. (So wie alles andere auch)

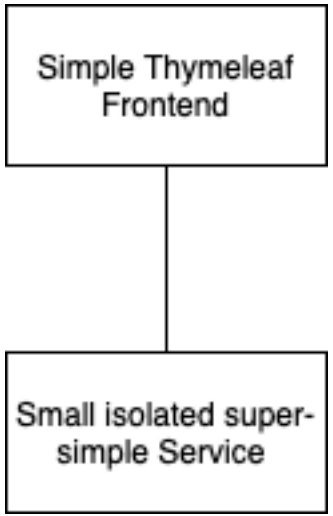
Research TODOs

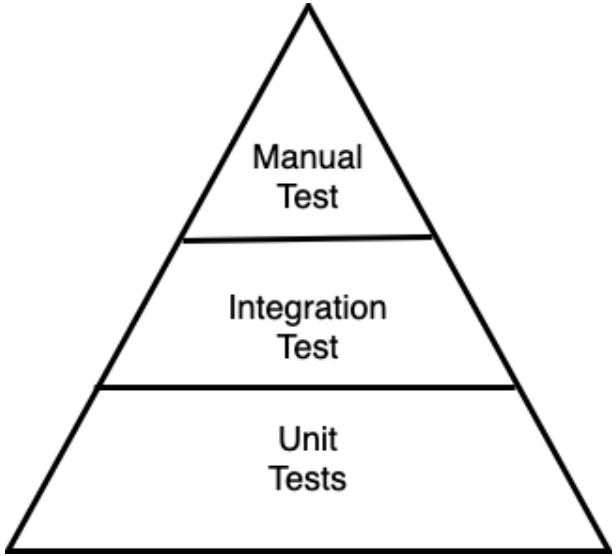
- https://en.wikipedia.org/wiki/Persistent_data_structure
- https://entwicklertag.de/karlsruhe/2014/sites/entwicklertag.de.karlsruhe.2014/files/folien/edit-vortrag/em-agiles-testen-handwerkszeug-zur-pr-vention-von-fehlern-und-technischen-schulden.pdf
- https://docs.pact.io/getting_started/testing-scope
- https://en.wikipedia.org/wiki/Extreme_programming

Ein paar Beispiele (1)



Ein paar Beispiele (1)



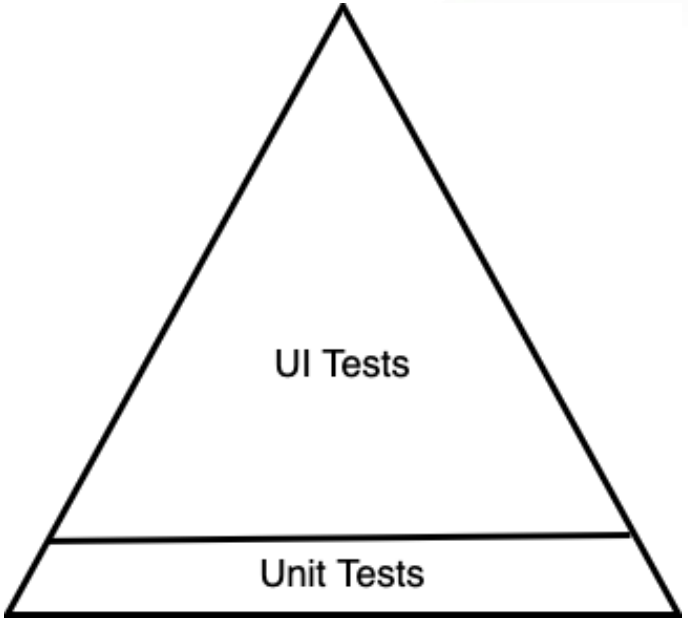


Ein paar Beispiele (2)

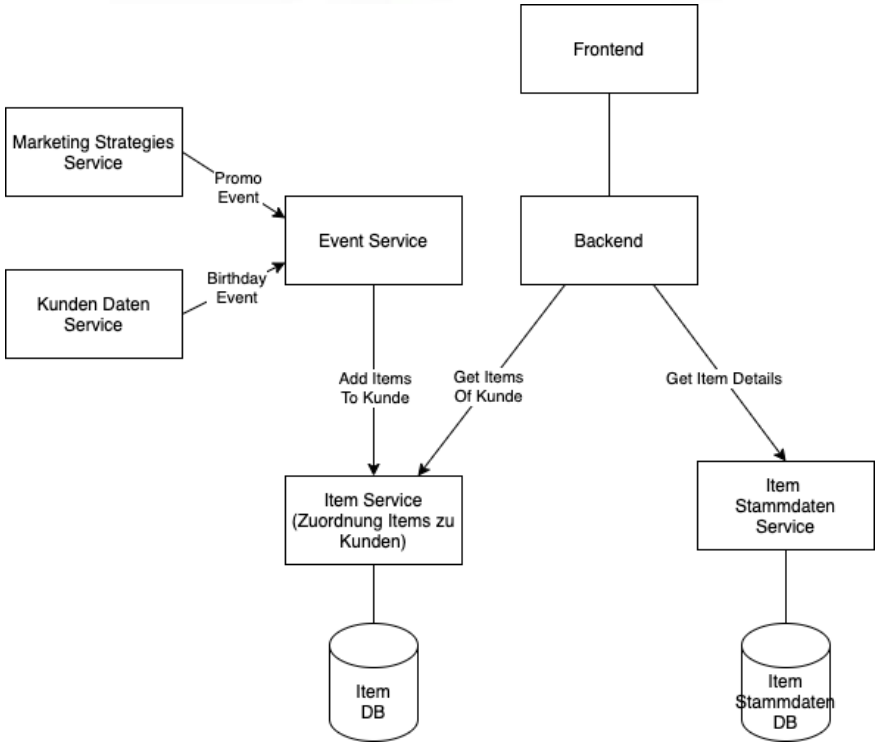
14-year old legacy monolith
without any tests or structure

Ein paar Beispiele (2)

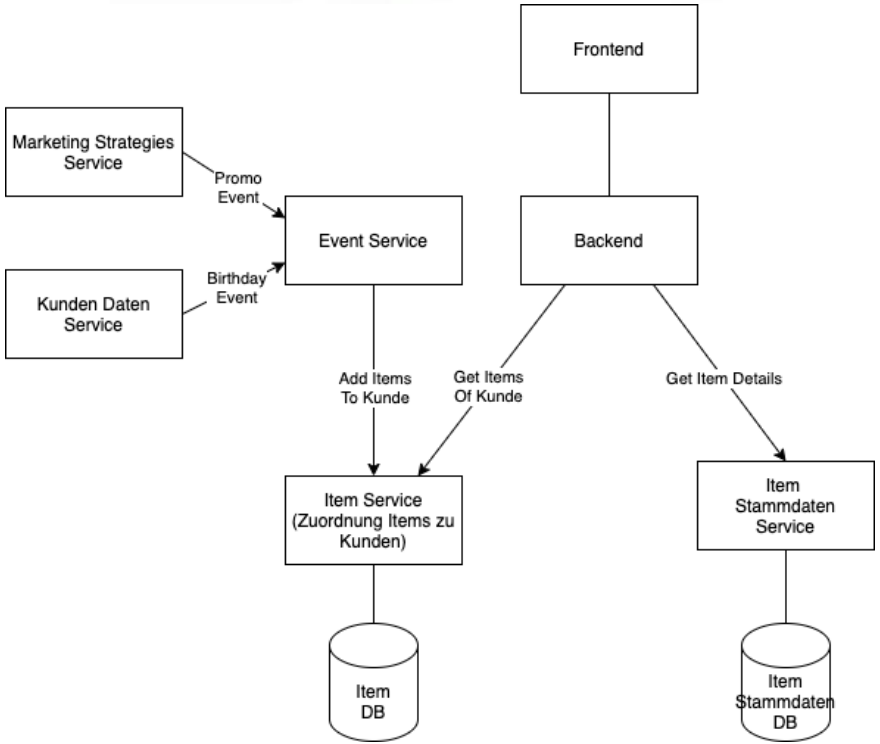
14-year old legacy monolith
without any tests or structure

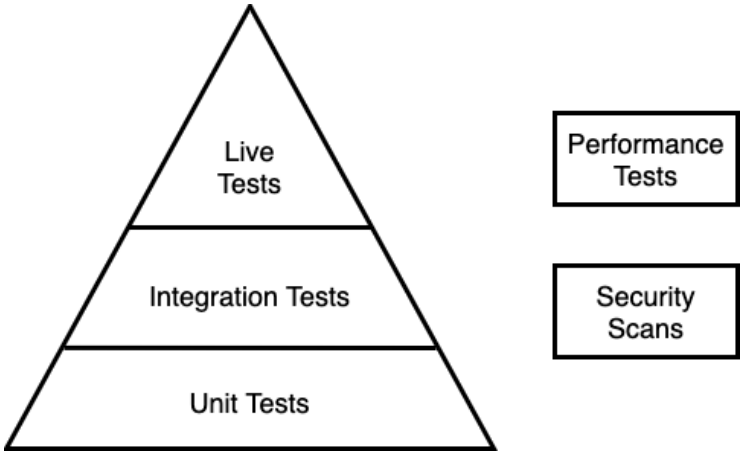


Ein paar Beispiele (3)



Ein paar Beispiele (3)

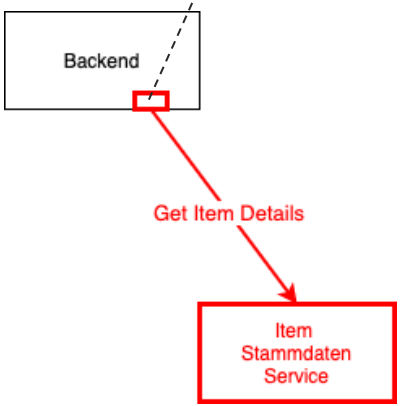




Contract Tests – Ein beliebter Pitfall

Contract Stub

Automatisch aus Contracts Generiert

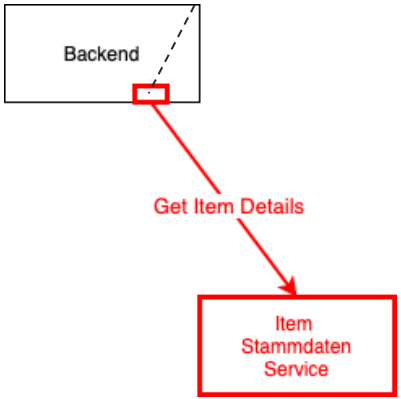


Contract Tests – Ein beliebter Pitfall

Contract Stub

Automatisch aus Contracts Generiert

(Quasi) Garantiert Korrekt

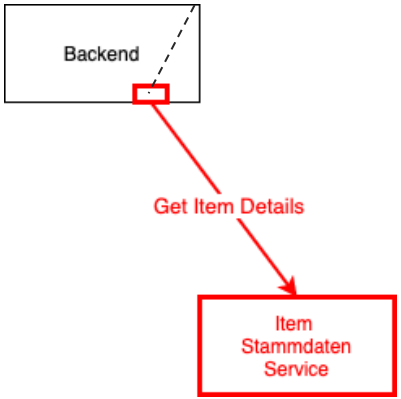


Contract Tests – Ein beliebter Pitfall

Contract Stub

Automatisch aus Contracts Generiert

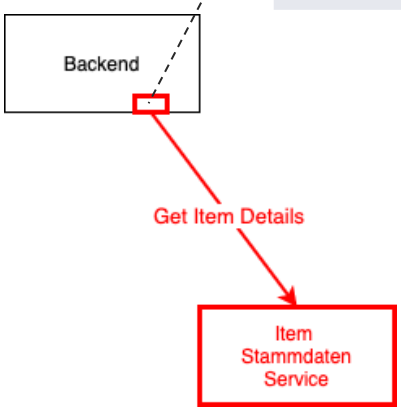
(Quasi) Garantiert Korrekt



Test Stub

Selbst geschrieben

Contract Tests – Ein beliebter Pitfall



Contract Stub

Automatisch aus Contracts Generiert

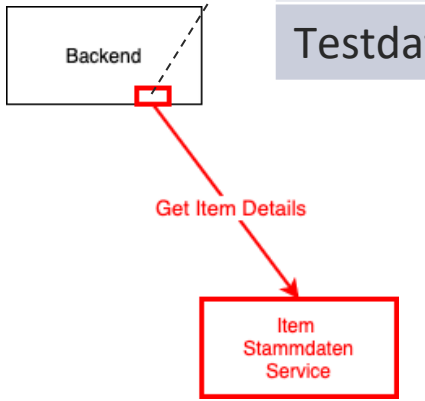
(Quasi) Garantiert Korrekt

Test Stub

Selbst geschrieben

Stubbed Verhalten kann überhaupt nicht mit echtem Verhalten korrelieren.

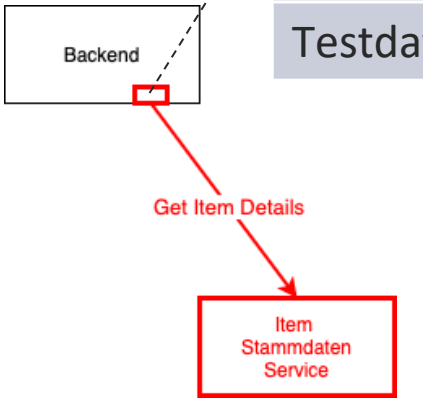
Contract Tests – Ein beliebter Pitfall



Contract Stub
Automatisch aus Contracts Generiert
(Quasi) Garantiert Korrekt
Testdaten in Contract definiert

Test Stub
Selbst geschrieben
Stubbed Verhalten kann überhaupt nicht mit echtem Verhalten korrelieren.
Testdaten selbst definiert

Contract Tests – Ein beliebter Pitfall

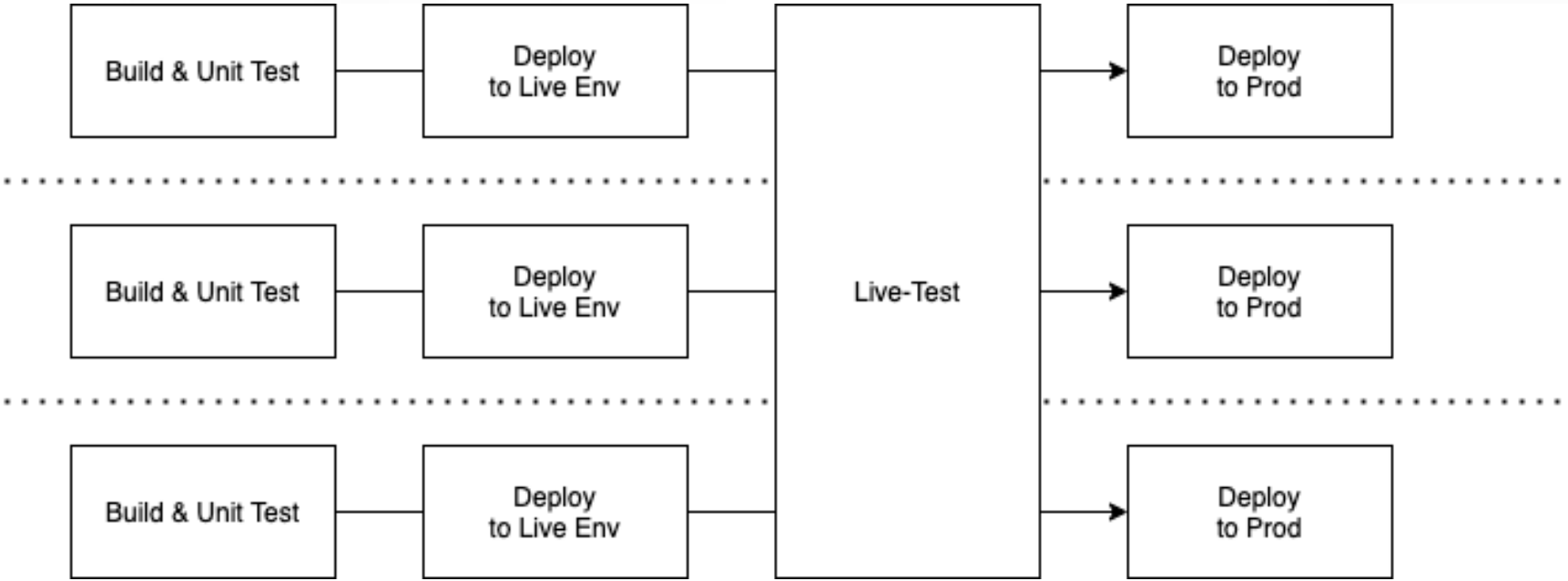


Contract Stub
Automatisch aus Contracts Generiert
(Quasi) Garantiert Korrekt
Testdaten in Contract definiert

Test Stub
Selbst geschrieben
Stubbed Verhalten kann überhaupt nicht mit echtem Verhalten korrelieren.
Testdaten selbst definiert

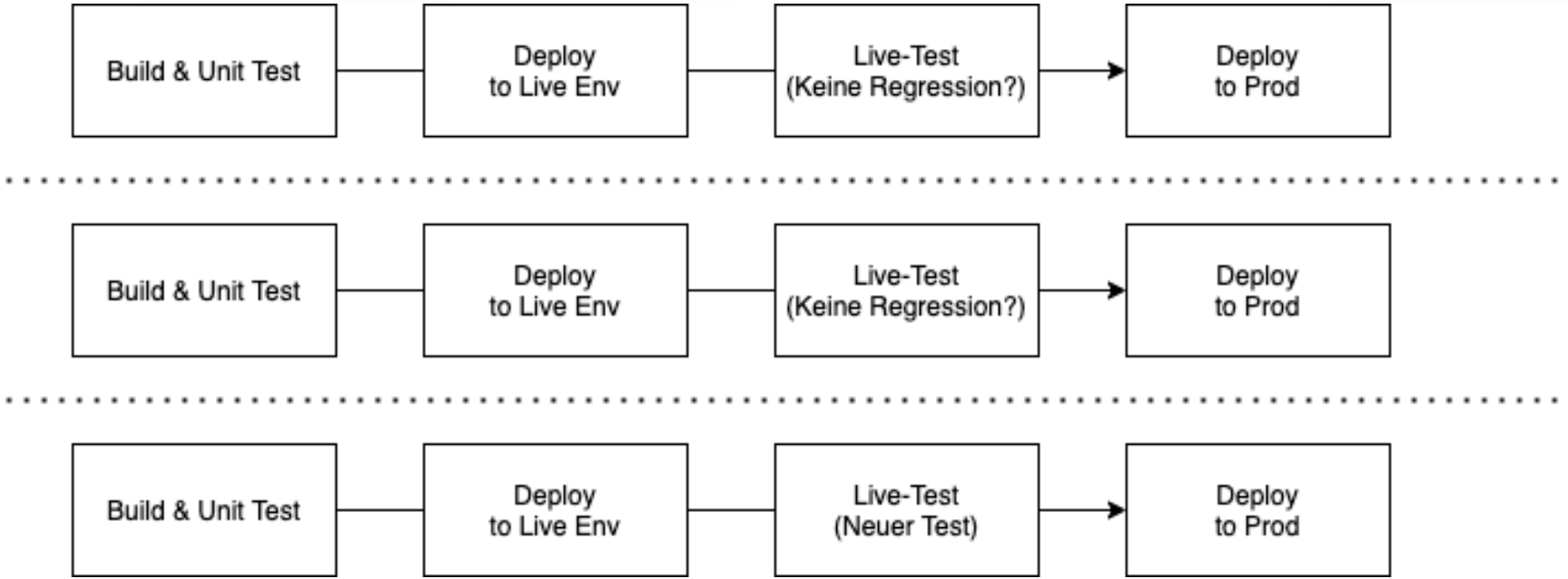
⇒ Benutzt Contract Stubs für Contract Tests.
⇒ Benutzt Test Stubs für alles Andere.

Live Tests – How to screw it up



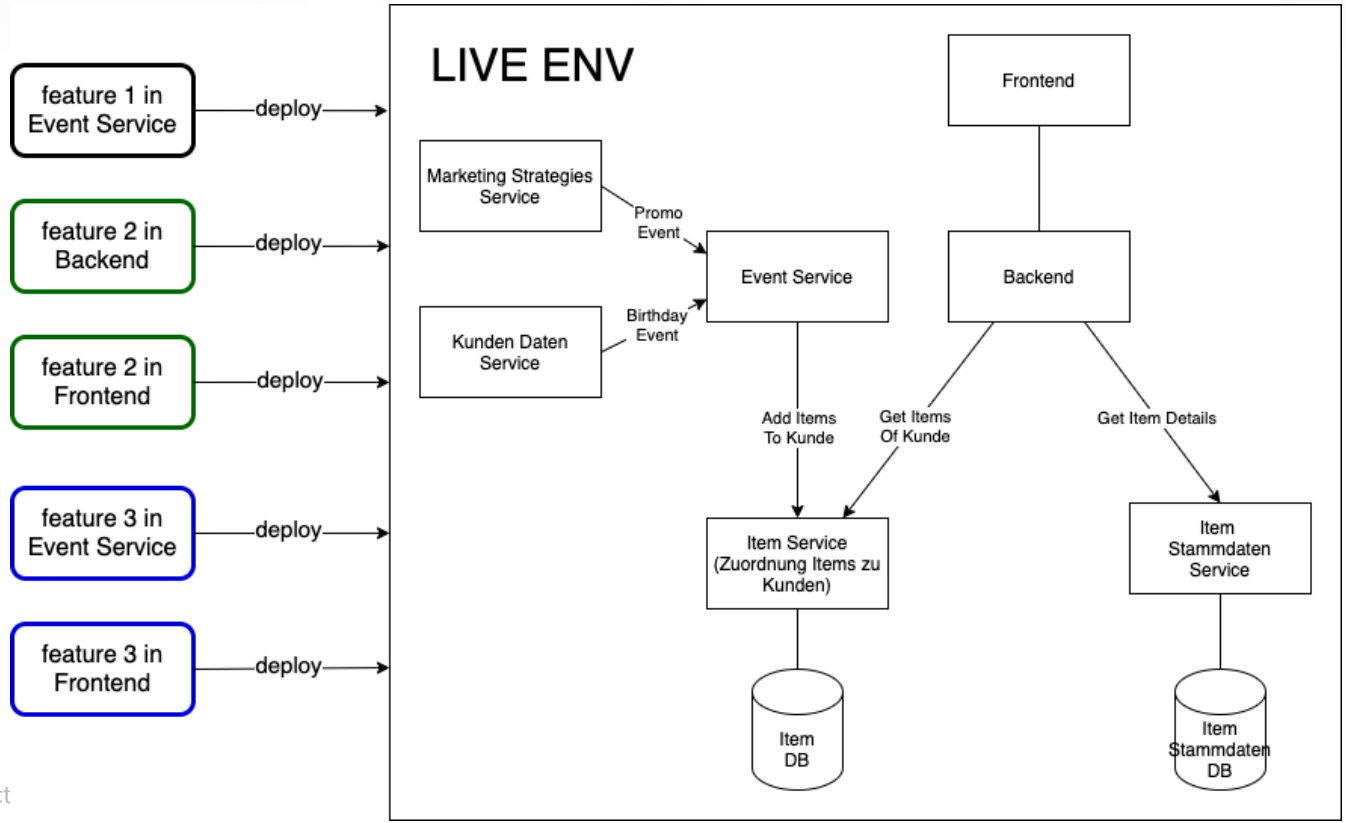
Auch bekannt als der „verteilte Monolith“

Live Tests – What it should look like



Sollte möglich sein, solange alle Services rückwärtskompatibel erweitert werden

Das Live Env – One for All



Das Live Env – One for All

- Billig zu runnen

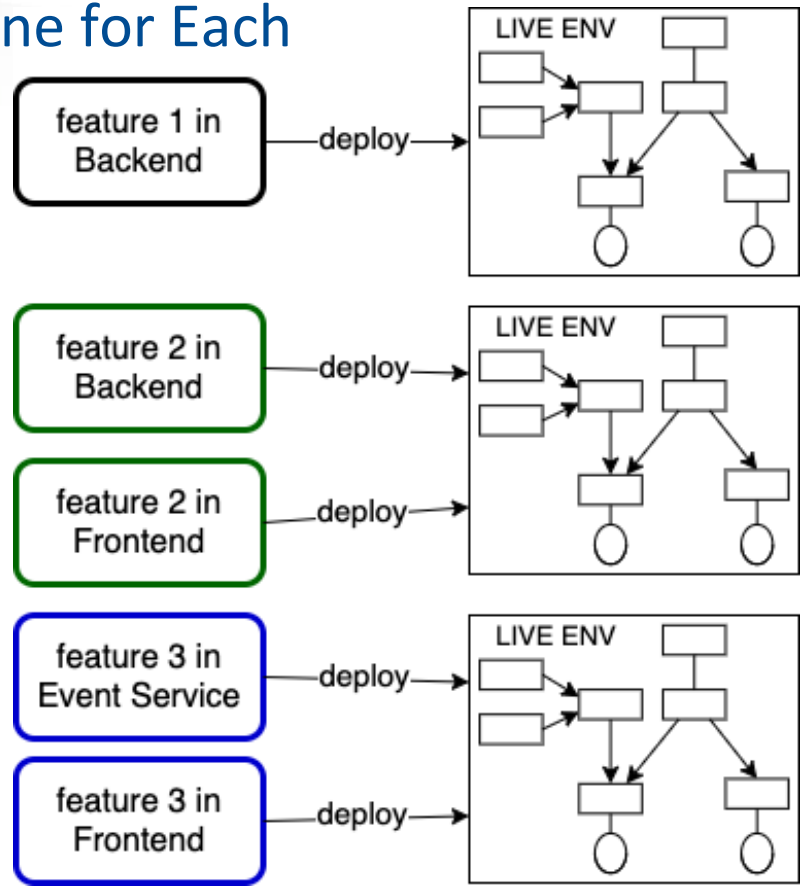
Das Live Env – One for All

- Billig zu runnen
- Einfach einzurichten

Das Live Env – One for All

- Billig zu runnen
- Einfach einzurichten
- Nebeneffekte durch andere Feature-Entwicklungen sind unmöglich zu eliminieren.

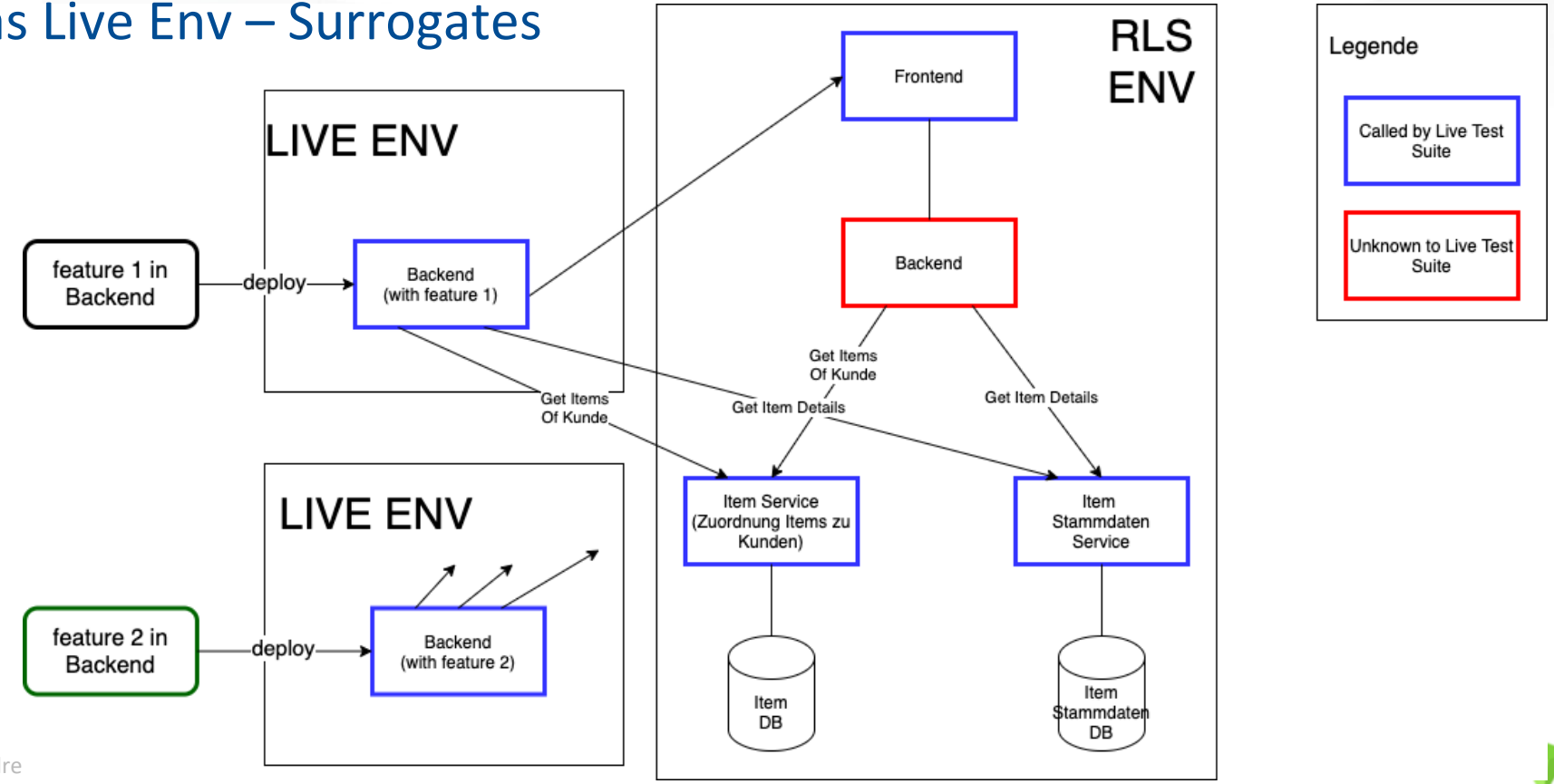
Das Live Env – One for Each



Das Live Env – One for Each

- Abhängig von wie groß das env ist, sauteuer zu runnen
- Aufwändig einzurichten
- Keine Nebeneffekte durch andere Feature-Entwicklungen

Das Live Env – Surrogates

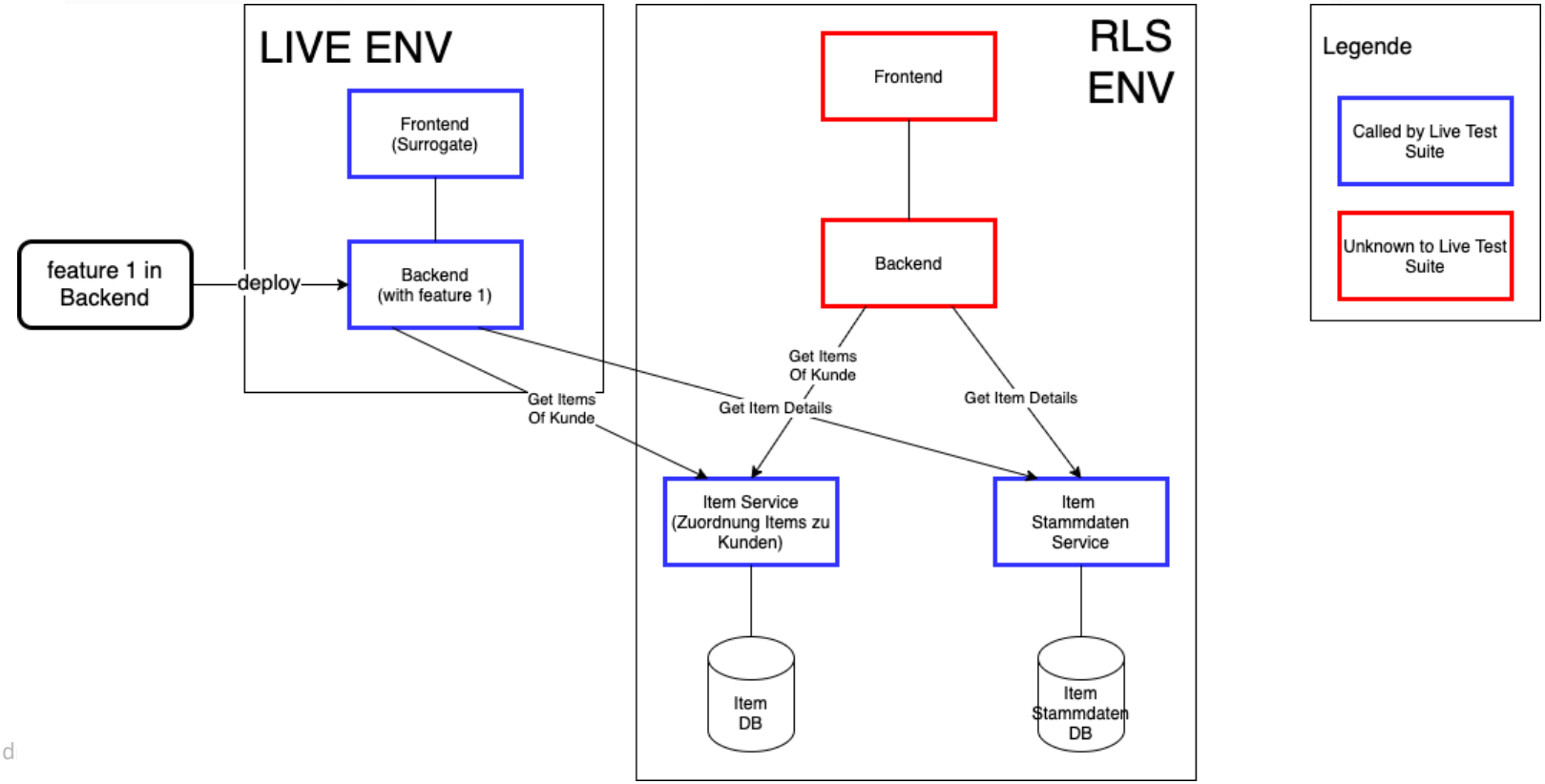


Das Live Env – Surrogates

Surrogate Definition:

Eine Instanz von dem master-Stand von einem Service, die konfiguriert ist um den „Service under Test“ anzusprechen.

Das Live Env – Surrogates



Das Live Env - Surrogates

- Marginal teurer als One for Each

Das Live Env - Surrogates

- Marginal teurer als One for Each
- Sehr aufwändig einzurichten

Das Live Env - Surrogates

- Marginal teurer als One for Each
- Sehr aufwändig einzurichten
- Keine Nebeneffekte zu anderen Feature Entwicklungen. (Wenn man nicht irgendwo etwas vergessen hat)

Das Live Env - Surrogates

- Marginal teurer als One for Each
- Sehr aufwändig einzurichten
- Keine Nebeneffekte zu anderen Feature Entwicklungen. (Wenn man nicht irgendwo etwas vergessen hat)
- Abhängig von der Architektur kann es passieren, dass die surrogates-Lösung nur marginal/gar nicht besser als „One for Each“ ist.

Das Live Env - Summary

	Cost to Run	Cost to Develop	Side Effects
One For All	$	$	
One For Each	$$$$$	$$	
Surrogates	$$	$$$	

* May vary, depending on your architecture

Das Live Env - Summary

	Cost to Run	Cost to Develop	Side Effects
One For All	$	$	
One For Each	$$$$$	$$	
Surrogates	$$	$$$	

* May vary, depending on your architecture

Es gibt noch viele andere Möglichkeiten & Hybrid-Lösungen

Das Live Env - Summary

	Cost to Run	Cost to Develop	Side Effects
One For All	$	$	
One For Each	$$$$$	$$	
Surrogates	$$	$$$	

* May vary, depending on your architecture

Es gibt noch viele andere Möglichkeiten & Hybrid-Lösungen

=> You‘re the engineers, figure it out.