

# Richtig testen in SOA/BPM-Projekten

Erfahrungen und Best Practices

Dipl.-Inform. Holger Breitling, Mitglied der Geschäftsleitung

Dipl.-Inform. Johannes Rost, Software-Architekt

24. September 2010



1. Kontext: SOA/BPM
2. Besondere Herausforderungen
3. Zielsetzung für BPM Tests
4. Projekthintergrund
5. Herausforderungen im Projekt
6. Lösungsansätze
7. Bewertung des Vorgehens



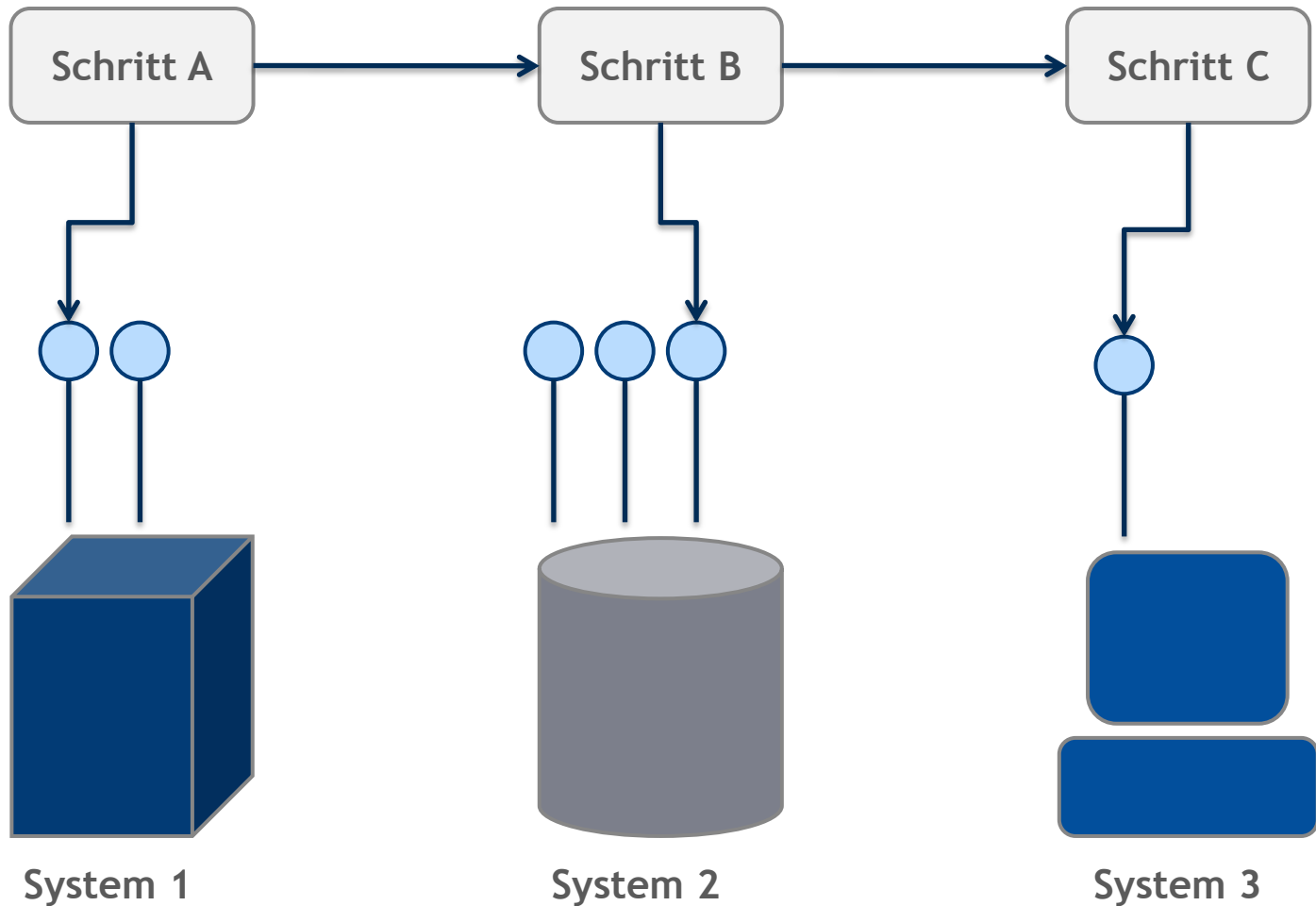
- **Service Oriented Architecture:**
  - Ermöglicht eine vereinfachte Integration bestehender Systeme
  - Wiederverwendung von (bestehender) Geschäftslogik
  - Flexiblere Kooperation
  
- **Business Process Management**
  - Automatisierte Geschäftsprozesse auf Basis von Services
  - Monitoring von Geschäftsprozessen
  - Geschäftsprozesse werden leichter anpassbar



- Prozess

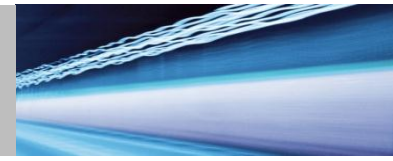
- Services (Operationen)

- Systeme

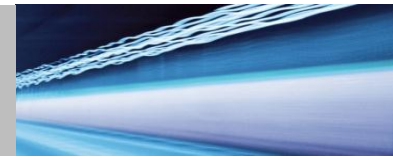




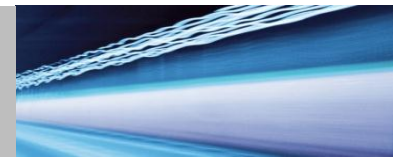
- Der Vortrag behandelt Tests, die mindestens eine Systemgrenze beinhalten.
- Normale Unit-Tests für die Implementierungen innerhalb einzelner Artefakte liegen nicht im Focus.



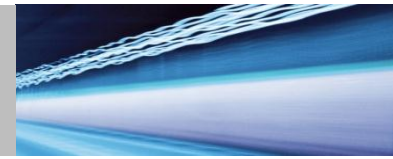
1. Kontext: SOA/BPM
2. Besondere Herausforderungen
3. Zielsetzung für BPM Tests
4. Projekthintergrund
5. Herausforderungen im Projekt
6. Lösungsansätze
7. Bewertung des Vorgehens



- Aufwändige Erzeugung von Testdaten
  - Häufig nur auf Basis von konsistenten Produktionsdaten
  - Datensätze in unterschiedlichen Systemen müssen zusammen passen.
  - Datensätze können unter Umständen nur einmal verwendet werden.
- Unterschiedliche Testkonzepte
  - Angebundene Systeme haben unterschiedliche Strukturen bei Testebenen (Entwicklung, Test, Integration, Produktion, ...)
  - Zentrale Testinstanzen vs. Mehrere Projekte (Ressourcenkonflikte)
- Unterschiedliche Wartungsfenster
  - Aufwändige Abstimmung gemeinsamer Testläufe



- Testdatensätze müssen in allen beteiligten Systemen bereitstehen
- Es muss eine gemeinsame Ebene für den Testlauf erstellt werden
- Zum Test passende Programmversionen müssen in allen Systemen installiert sein
- Es muss ein gemeinsamer Termin für den Testlauf gefunden werden
- Nach der Korrektur von gefundenen Fehlern geht alles wieder von vorne los  
=> Derartige Tests sind sehr teuer!



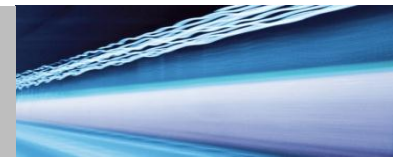
1. Kontext: SOA/BPM
2. Besondere Herausforderungen
3. Zielsetzung für BPM Tests
4. Projekthintergrund
5. Herausforderungen im Projekt
6. Lösungsansätze
7. Bewertung des Vorgehens



- Möglichst große Teile der Gesamtlösung testen, ohne aufwändige Integrationstests des gesamten Setups zu benötigen.
- Vollautomatische Regressionstests für möglichst große Teile der Gesamtlösung.
- Benötigte Anzahl von Test-Korrektur-Zyklen auf Integrationstest-Ebene gering halten.

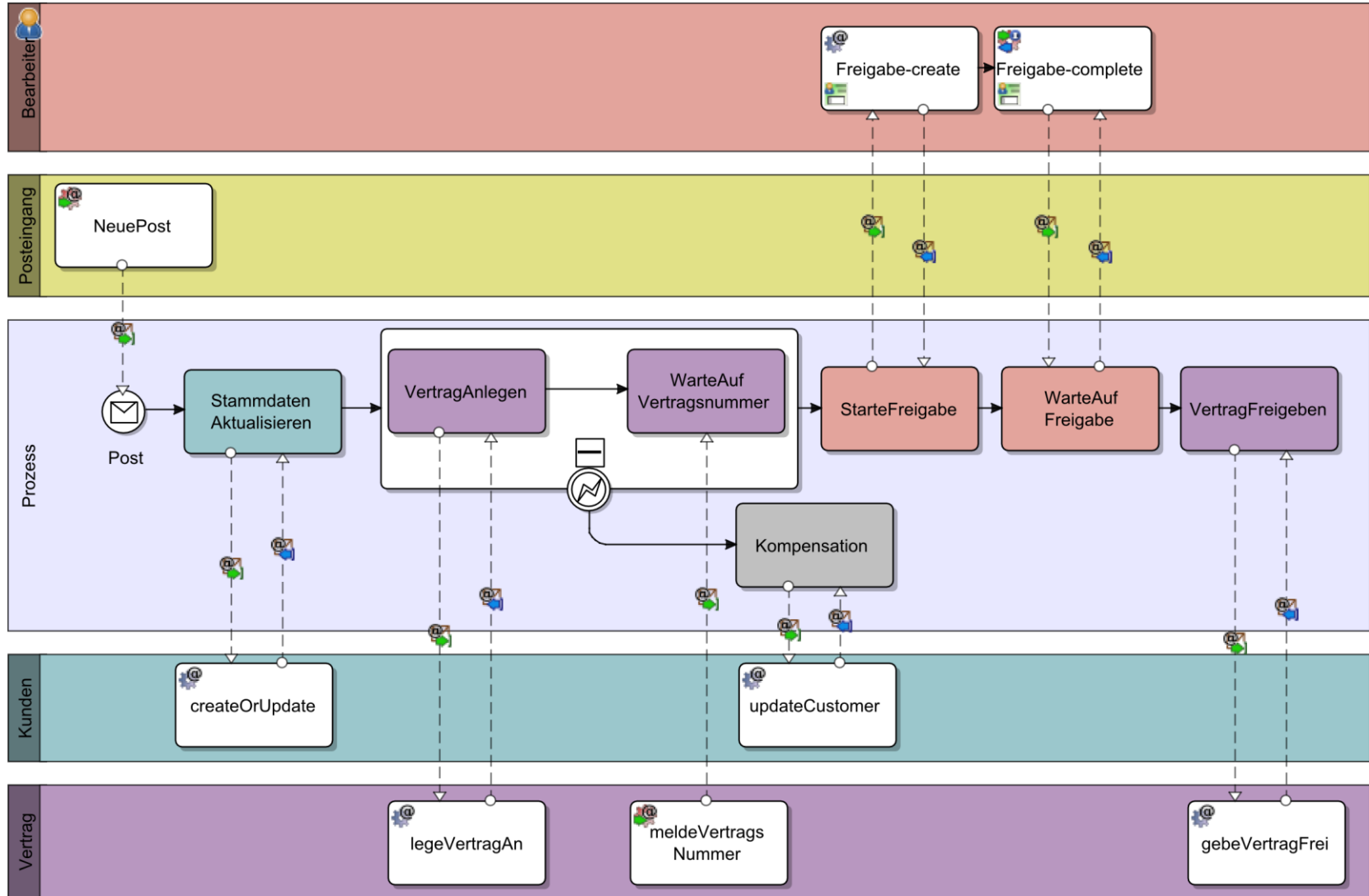


1. Kontext: SOA/BPM
2. Besondere Herausforderungen
3. Zielsetzung für BPM Tests
4. Projekthintergrund
5. Herausforderungen im Projekt
6. Lösungsansätze
7. Bewertung des Vorgehens



- Kunde: Eine deutsche Versicherung
- Einführung und Weiterentwicklung einer SOA auf Basis von Sopera ASF
- Implementierung von automatisierten Geschäftsprozessen in Intalio auf Basis der SOA
- Flexible Reaktion auf sich ändernde Anforderungen im Projektverlauf wichtig, da sich Anforderungsermittlung und Implementierung überschneiden.
  - Automatisierte Regressionstests auf allen Ebenen.

# Beispielprozess (vereinfacht)





1. Kontext: SOA/BPM
2. Besondere Herausforderungen
3. Zielsetzung für BPM Tests
4. Projekthintergrund
5. Herausforderungen im Projekt
6. Lösungsansätze
7. Bewertung des Vorgehens



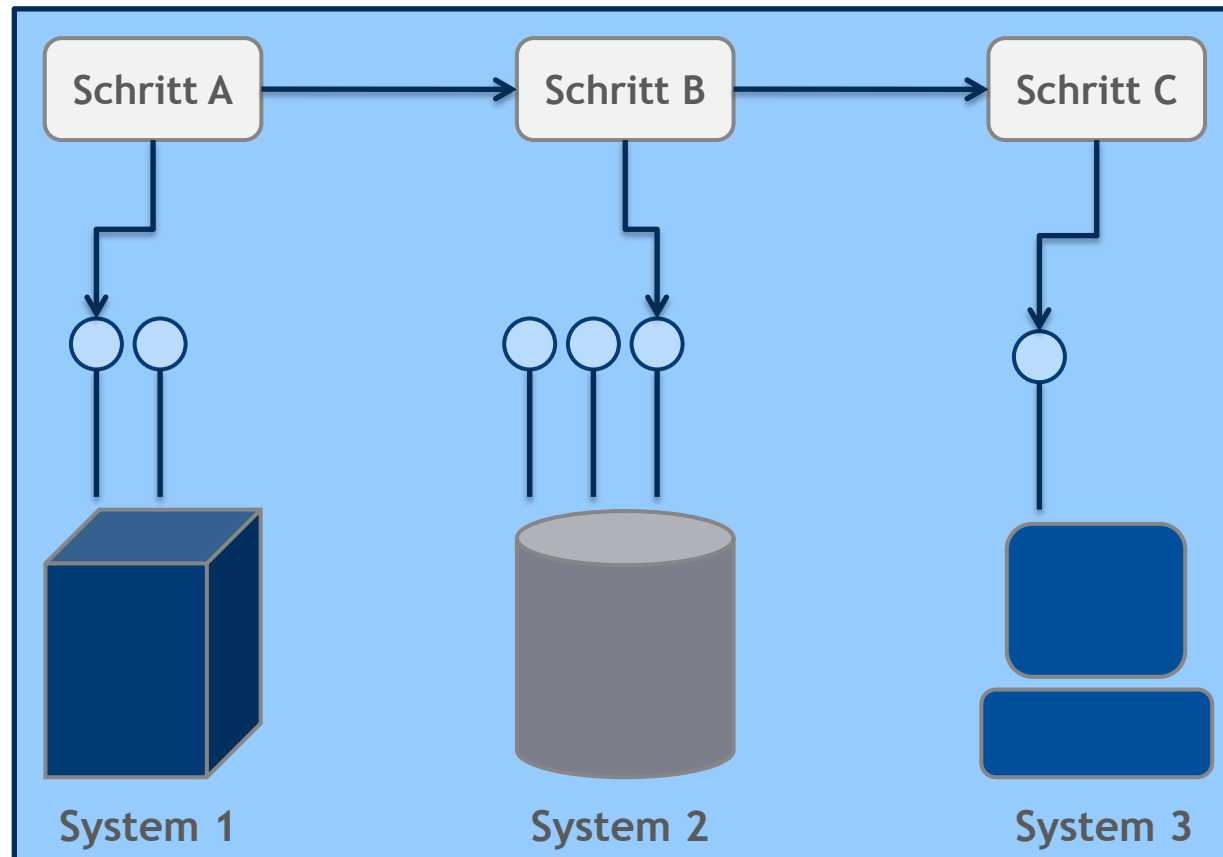
- Schon am Pilotprozess sind Systeme mit sehr unterschiedlichen Architekturen beteiligt
- Testdatensätze, die in allen Systemen funktionieren, sind nur manuell erstellbar
- Inhalt der Datensätze unterscheidet sich von System zu System
  - Fehlerbehandlung bei abgelehnten Datensätzen wichtig
- Die verwendete SOA/BPM-Suite bringt bisher nur rudimentäre Werkzeuge zur Unterstützung von Tests mit.



1. Kontext: SOA/BPM
2. Besondere Herausforderungen
3. Zielsetzung für BPM Tests
4. Projekthintergrund
5. Herausforderungen im Projekt
6. Lösungsansätze
7. Bewertung des Vorgehens



- Da der Aufwand für vollständige Integrationstests sehr hoch ist, mussten andere Varianten zum Testen gefunden werden.

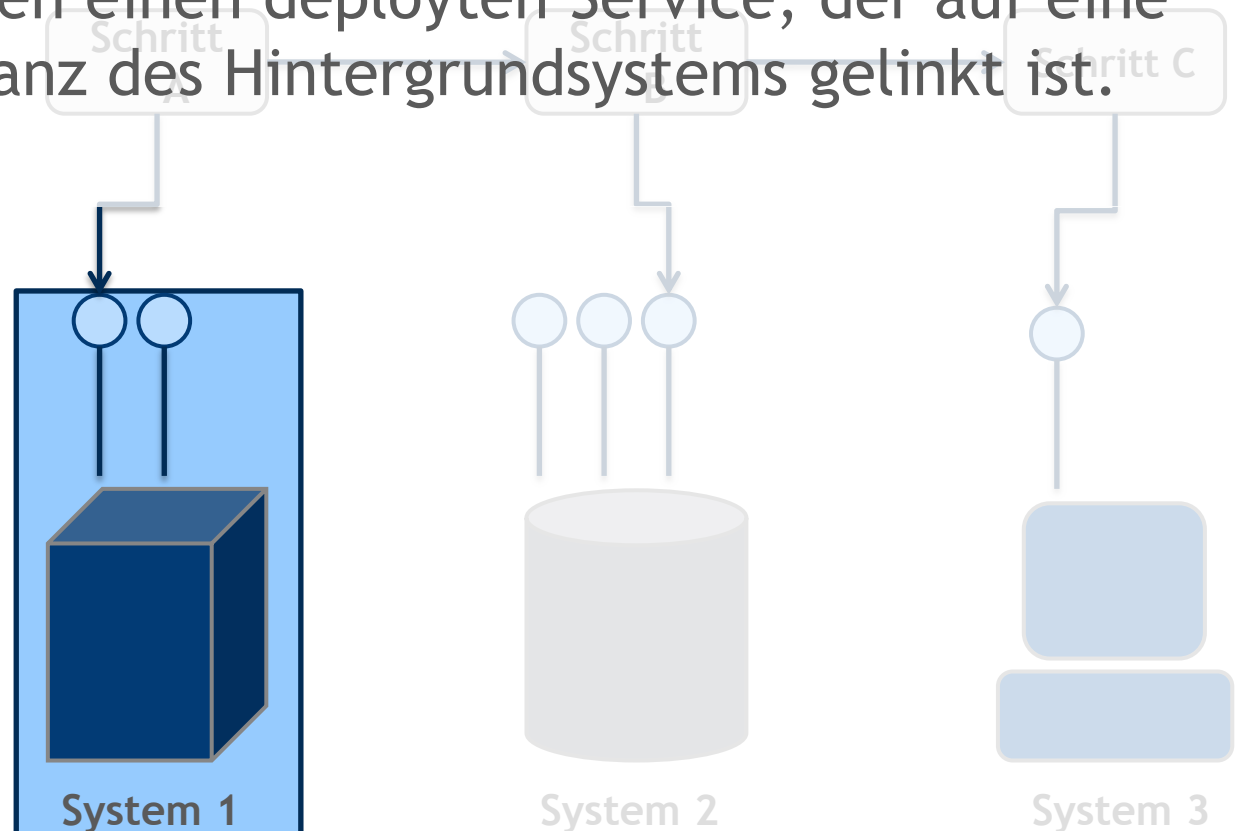


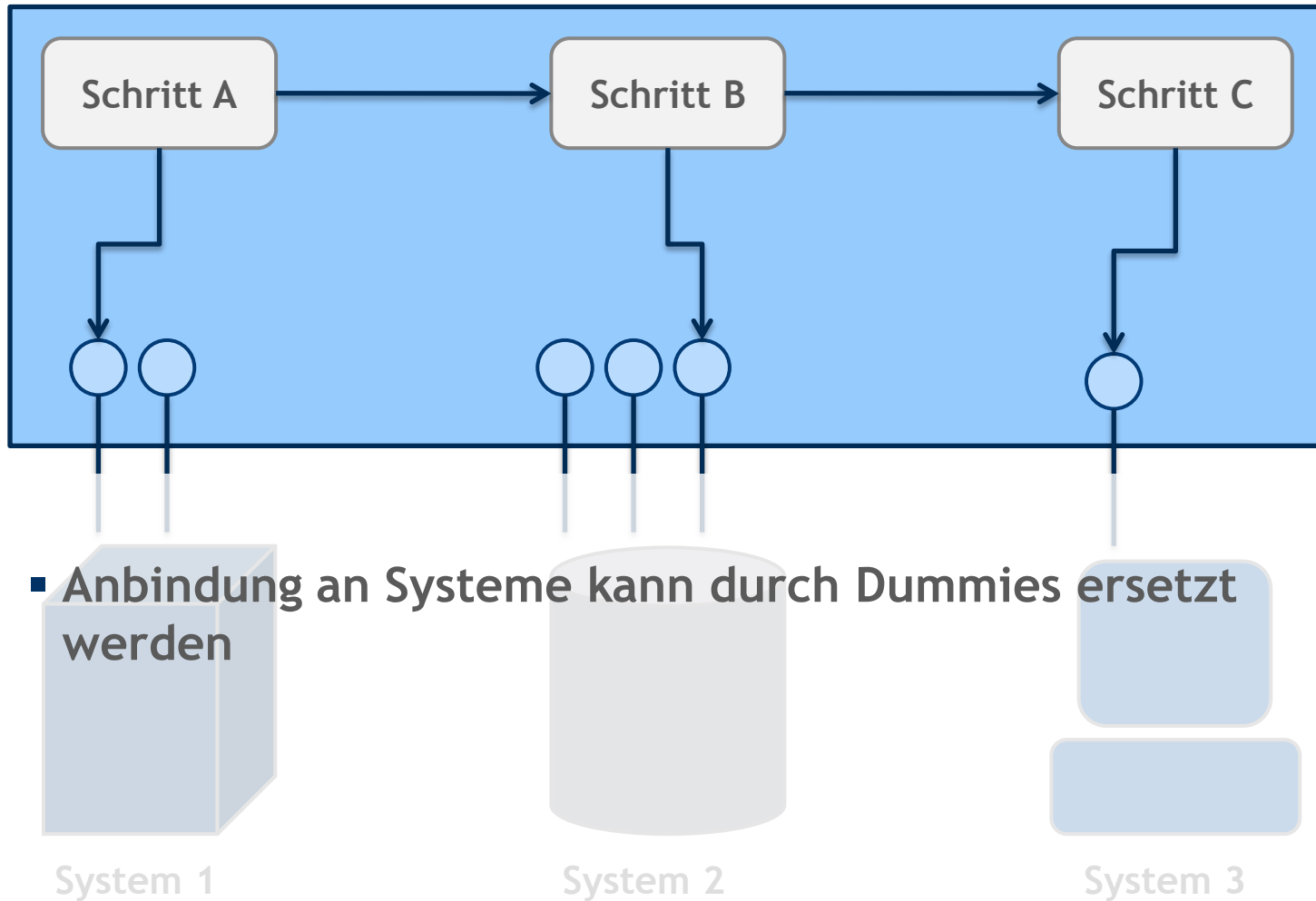


- Prozesstests, Servicetests und mehrere Stufen von Integrationstests voneinander trennen
- Services können einzeln gegen ihre definierte Schnittstelle getestet werden: Von diesen Tests sind nur wenige Systeme gleichzeitig betroffen.
- Interne Tests der Serviceimplementierung (Java) auf Basis von JUnit
- Die Prozesse werden unabhängig von Serviceimplementierungen getestet.
- Interne Unit-Tests für Prozesse nicht möglich, da die verwendete BPEL-  
Runtime dazu nicht ausgelegt ist.



- Automatisierte Tests einzelner Services können die korrekte Anbindung des Hintergrundsystems testen.
- Der Test läuft gegen einen deployten Service, der auf eine passende Testinstanz des Hintergrundsystems gelinkt ist.



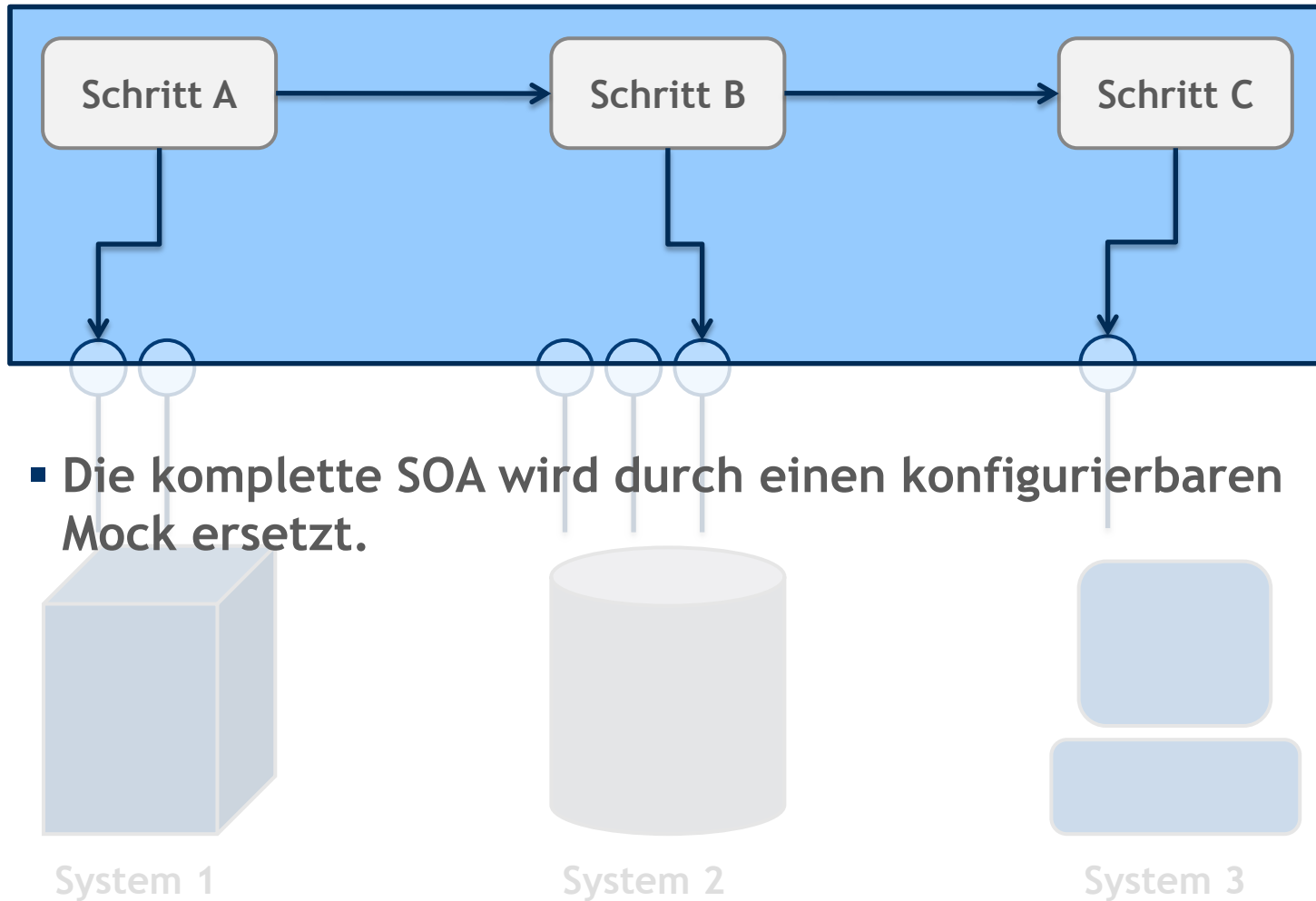




- Services können neben der vollen Implementierung auf Basis von Hintergrundsystemen noch eine Testimplementierung bereitstellen
- Dadurch werden die Abhängigkeiten zum Zustand der Hintergrundsysteme entfernt
- Diese Tests können zum Testen von Erreichbarkeit von Services, korrekter Anbindung an den Prozess, Berechtigungen, usw. genutzt werden.

Zu beachten:

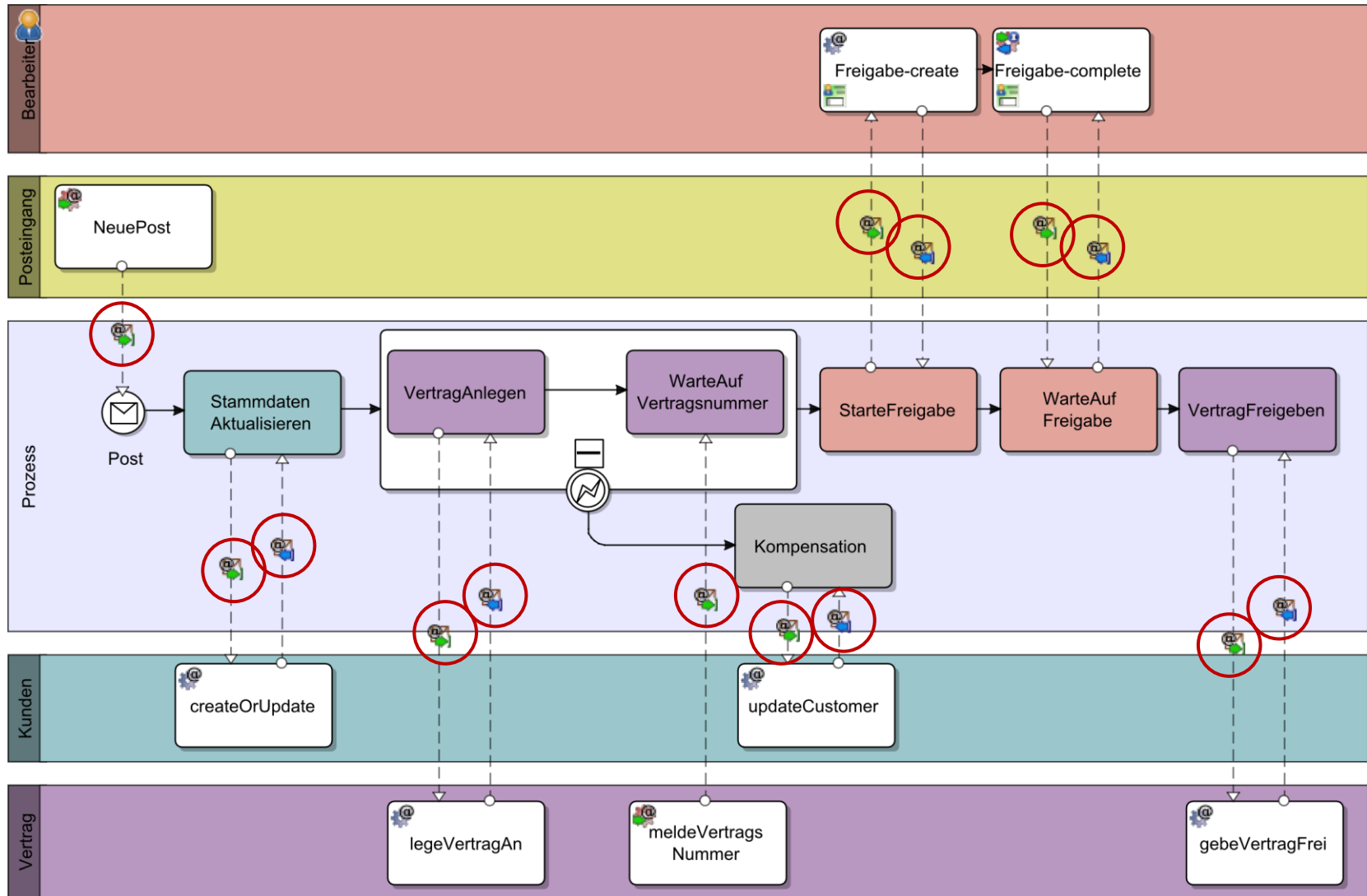
- Es entsteht zusätzlicher Aufwand für die Testimplementierung
- Das Setup für solche Integrationstests ist immer noch komplex (z.B. Deployment der Services)





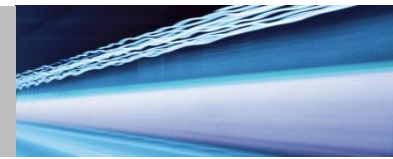
- Die verwendete SOA/BPM-Suite bringt keine direkte Unterstützung für derartige Testfälle mit.
- Die offene Architektur erlaubt eine Eigenentwicklung mit geringem Aufwand:
  - Implementierung in Java (Sopra-Libraries sind vorhanden, gutes XML-Handling, Implementierungssprache für die Services)
  - JUnit wird als Testrunner verwendet (Gute IDE-Integration, gute Integration mit weiteren Frameworks)
  - Das Spring Framework wird für Konfiguration und Setup verwendet.

# Basis für den Test: Ausgetauschte Nachrichten



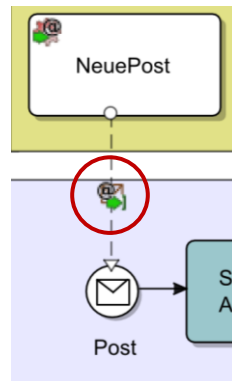


- Ein zentraler, konfigurierbarer Mock wird als Implementierung für alle benötigten Services angemeldet.
- Ein Testfall besteht aus einer Konfiguration für diesen Mock und aus Code, der den Ablauf steuert.
- Jede Nachricht, die an den Prozess geschickt wird, wird als XML-Datei abgelegt.
- Der Testcode übernimmt das Senden von Nachrichten an den Prozess und das Annehmen von Service-Calls vom Prozess.



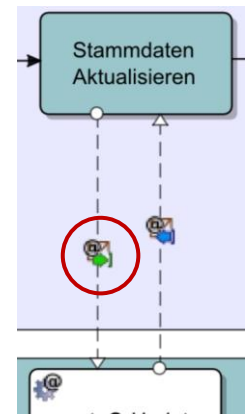
## Schritt 1

- Startnachricht an den Prozess schicken



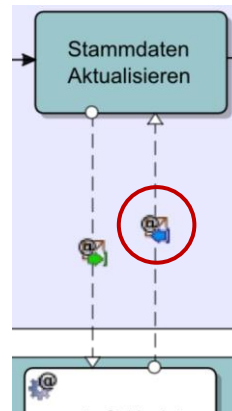
## Schritt 2

- Serviceaufruf „Stammdaten aktualisieren“ entgegennehmen



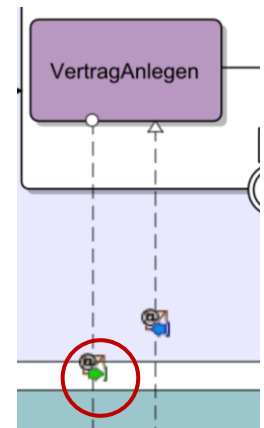
## Schritt 3

- Serviceaufruf „Stammdaten aktualisieren“ beantworten



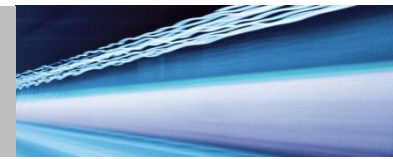
## Schritt 4

- Serviceaufruf „Vertrag anlegen“ entgegennehmen





- Im Verlauf des Testfalls wird für jede empfangene und gesendete Nachricht ein Log-Eintrag erzeugt.
- Am Ende des Testfalls werden die Log-Einträge auf korrekte Anzahl und Reihenfolge geprüft.
- Durch einen Test auf dieser Ebene lässt sich der Kontrollfluss eines Prozesses testen.
  
- Zu beachten: Bisher wird der Inhalt der Nachrichten nicht berücksichtigt.



- Der erwartete Inhalt für vom Prozess gesendete Nachrichten wird ebenfalls in der Konfiguration des Testfalls festgelegt.
- Die Vorbedingungen lassen sich als XPath-Ausdrücke formulieren, z.B. `/Brief/Absender/Name=„Müller“`
- Um die Testfälle flexibler zu gestalten, werden Variablen eingeführt, die in der Konfiguration verwendet und vom Testfall gesetzt werden:

## Nachricht:

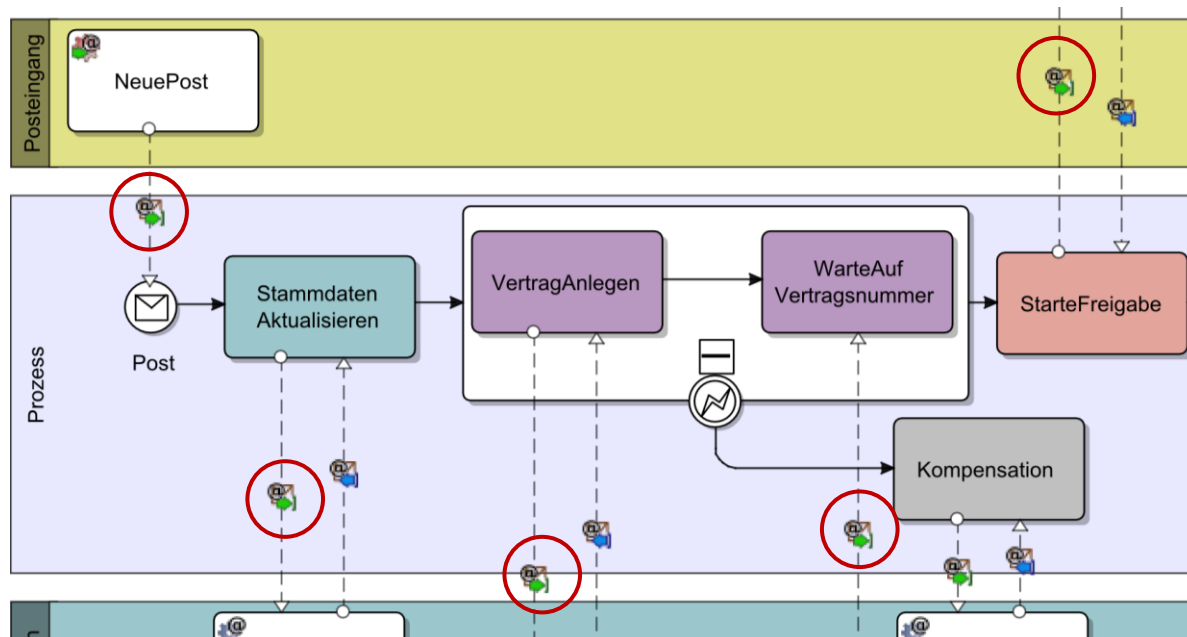
```
<Brief>
  <Absender>
    <Name>${name}</Name>
  ...
```

## Vorbedingung:

```
/Brief/Absender/Name=„${name}“
```

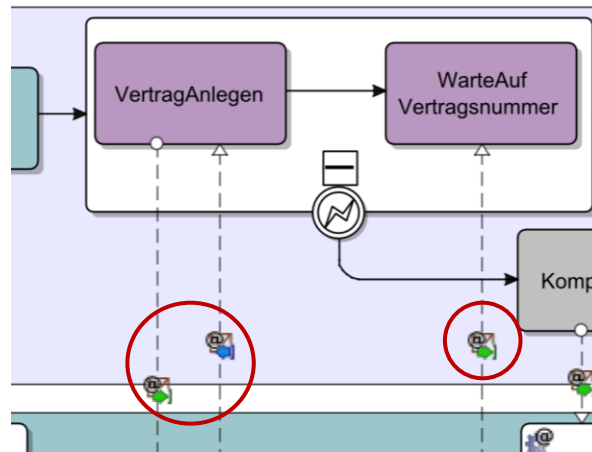


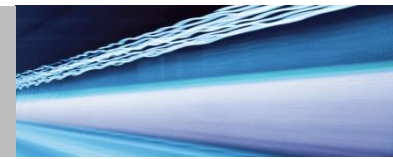
- Der Testfall stellt so sicher, dass an allen Stellen derselbe, richtige Name auftaucht.
- Unterschiedliche Testfälle können auf denselben Nachrichten und Vorbedingungen basieren, aber unterschiedliche konkrete Werte verwenden.



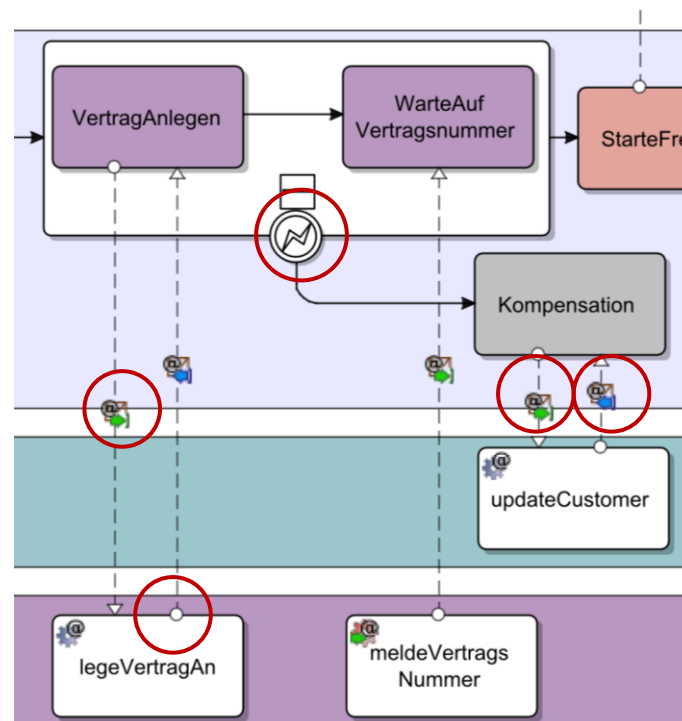


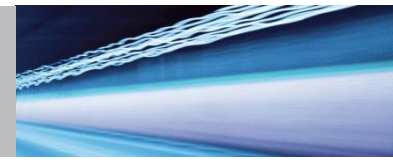
- Durch den Test des Nachrichteninhaltes kann das Mapping von Daten im Prozess getestet werden.
- Damit können neben dem Kontrollfluss weitere Implementierungsdetails getestet werden.
- Die verwendeten Variablen ermöglichen auch den Test asynchroner Aufrufe, für die der getestete Prozess Nachrichteninhalte und Prozessinstanzen korrelieren muss.





- Das Testframework ist in der Lage, auf einen Aufruf mit einem Fehler anstelle einer Nachricht zu antworten.
- Dadurch lässt sich auch die Reaktion eines Prozesses auf eine Fehlersituation und der Aufruf der entsprechenden Kompensation testen.





- Der Intalio BPM-Server bietet eine Webservice-API zur Abfrage von Prozessinstanzen.
- Die Abfragen können mit Filtern zu beliebigen Eigenschaften von Prozessinstanzen versehen werden.
- Dadurch kann nach jedem Testfall überprüft werden, ob die erwarteten Prozessinstanzen im erwarteten Zustand im Server vorhanden sind.
- So kann z.B. sichergestellt werden, dass eine Prozessinstanz tatsächlich „Completed“ ist und nicht nach dem Testfall hängenbleibt.



- Für Lasttests wurde das Testframework so erweitert, dass Trigger für asynchrone Aufrufe an beliebige Operationsaufrufe gebunden werden können.
- Für Operationsaufrufe können zufällige Antwortzeiten konfiguriert werden.
- Eine festgelegte Anzahl Prozessinstanzen wird in einer definierten Zeitspanne zufällig gestartet.
- Der Lasttest mit zufälligen Zeiten führte schon bei deutlich geringerem Durchsatz zu Fehlersituationen.
- Die Lasttests mit dem Framework konnten trotz Ressourcenengpässen in den Testinstanzen der Hintergrundsysteme zuverlässig durchgeführt werden.



1. Kontext: SOA/BPM
2. Besondere Herausforderungen
3. Zielsetzung für BPM Tests
4. Projekthintergrund
5. Herausforderungen im Projekt
6. Lösungsansätze
7. Bewertung des Vorgehens



- Durch die strikte Aufteilung der Prozess-Testfälle in Nachrichteninhalte, Konfiguration und Testlogik ließen sich die einzelnen Bestandteile der Testfälle gut wiederverwenden.
  - Dadurch wurde erheblicher Testerstellungsaufwand eingespart.
- Durch die gründlichen Komponententests konnte erreicht werden, dass im viel aufwändigeren Integrationstest so gut wie keine Fehler mehr auftraten
  - Das bedeutet: Die Komponententests waren effektiv und führten frühzeitig zur Beseitigung von Fehlern und zu verbesserter Softwarequalität
  - Der Aufwand für Fehlernachtests auf Integrationsebene konnte minimiert werden – damit auch der Gesamttestaufwand.



- Die Lasttests identifizierten Fehler in der Prozess-Engine im Zusammenspiel mit der verwendeten Datenbank (DB2).
- Diese Fehler wären ohne die dedizierten Prozess-Lasttests vermutlich erst in der Produktionsumgebung aufgetreten, da die Testinstanzen der Hintergrundsysteme stark ausgebremst sind.
- Die Lasttests konnten als vollständige Konfiguration an den Hersteller der Prozess-Engine geschickt werden.
- Der Hersteller der Prozess-Engine konnte so rechtzeitig Fixes für die gefundenen Probleme bereitstellen.

Danke für Ihre Aufmerksamkeit!



# Haben Sie Fragen

