

Service-orientierte Software-Architekturen

Prof. Dr. U. Hoffmann
FH Wedel

Service–Lebenszyklus und Versionierung

Muster des Nachrichtenaustauschs (Message Exchange Patterns)

- ▶ Basis MEPs
 - ▶ Anfrage/Antwort-Muster
 - ▶ Einweg-Nachrichten-Muster
(2 x Einweg vs. Anfrage/Antwort)
- ▶ komplizierte MPEs
 - ▶ Anfrage-Rückruf-Muster
 - ▶ Veröffentlichen/Abbonieren-Muster
 - ▶ Behandlung von Fehlern
- ▶ MEP auf verschiedenen Ebenen
- ▶ Ereignisgesteuerte Architektur

Service- Lebenszyklus und Versionierung

Lebenszyklus

Versionierung

Fachlich getrieben

Datentypen

Konfigurationsmanagement

Ausblick

Service–Lebenszyklus

Versionierung

Fachlich getriebene Versionierung
Versionierung von Datentypen
Konfigurationsmanagement

Ausblick

Service– Lebenszyklus und Versionierung

Lebenszyklus

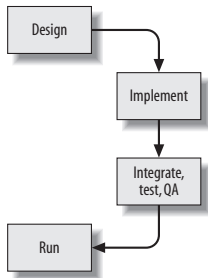
Versionierung

Fachlich getrieben

Datentypen

Konfigurationsmanagement

Ausblick



Service– Lebenszyklus und Versionierung

Lebenszyklus

Versionierung

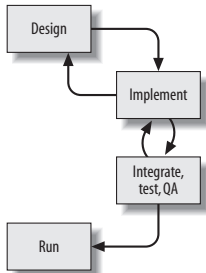
Fachlich getrieben
Datentypen
Konfigurationsmanagement

Ausblick

Services

- ▶ sind Software–Repräsentationen fachlicher Funktionalität
- ▶ unterliegen wie jede Software einem Lebenszyklus
- ▶ klassische Phasen des Software–Engineering

Graphik: *SOA in Practive*, N. Josuttis



Service– Lebenszyklus und Versionierung

Lebenszyklus

Versionierung

Fachlich getrieben

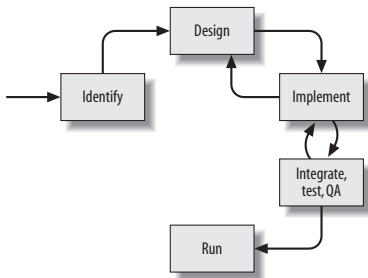
Datentypen

Konfigurationsmanagement

Ausblick

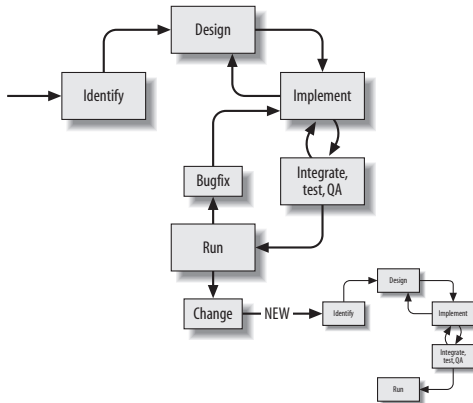
Inkrementelle, iterative Service–Entwicklung

- ▶ Service ist Teil größerer Geschäftsprozesse
- ▶ Änderung des Services hat Auswirkungen auf andere Systeme
- ▶ Design: Definition der Schnittstelle
syntaktisch, semantisch, nichtfunktionale Aspekte
- ▶ Schnittstellenänderungen als Normalfall akzeptieren
- ▶ Schnittstellenänderungen planen (z.B. als Vertragsbestandteil)
- ▶ Rückkopplung zwischen Test, Implementierung und Design



Identifizieren von Services (Entdecken, klassisch: Analyse-Phase)

- ▶ Geschäftsprozessmodellierung, Solution–Design
- ▶ Portfoliomanagement
- ▶ geänderte (neue oder zusätzliche) Anforderungen
- ▶ neue Service–Version: neuer Service mit neuem Lebenszyklus



Service geht in Betrieb (in Produktion)

- Einspielen (Deployment) im Laufenden (Produktiv–)System

Änderung eines im Betrieb befindlichen Services

- Schnittstellenänderung, neue Funktionalität:
Verhaltensänderung: neue Service–Version
- Fehlerkorrektur: *Bugfix* in bestehender Version

Service–
Lebenszyklus
und
Versionierung

Lebenszyklus

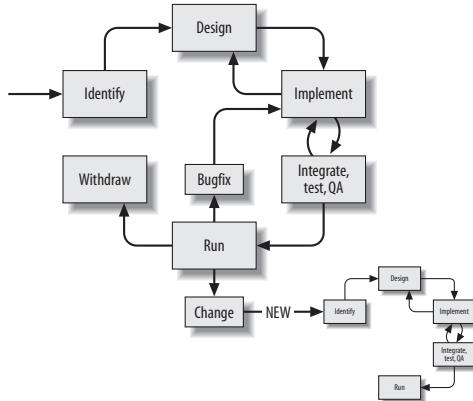
Versionierung

Fachlich getrieben

Datentypen

Konfigurationsmanagement

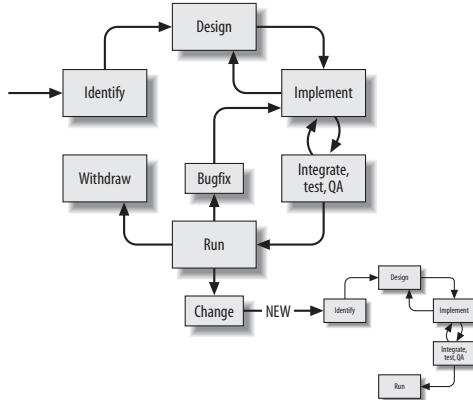
Ausblick



Graphik: *SOA in Practive*, N. Josuttis

Außerbetriebnahme (Withdrawal) von Services

- ▶ Außerbetriebnahme für übersichtliche SOA–Landschaft nötig
- ▶ alle Abhängigkeiten müssen erkannt und gelöst werden
- ▶ Außerbetriebnahme liefert keine neue Funktionalität
- ▶ daher typischerweise mit niedriger Priorität verfolgt
- ▶ Übergangsphase mit mildem Druck durch Anbieter nötig



Möglicher Prozess für die Außerbetriebnahme:

1. Service als veraltet (*deprecated*) markieren, Alternativen und Migrationsstrategie anbieten
2. Beobachten, welche Systeme den Service verwenden (ESB–Protokollierungsmöglichkeit)
3. Nach Übergangszeit: Verhandlung mit Nutzern über Ausstiegsplan

Änderung der Schnittstelle von Services

- ▶ Aufgrund sich ändernder Anforderungen für Produktiv–Services
- ▶ während der Umsetzung als Designänderung
 - ▶ Während der Entwicklung handhabbar (führt nicht zu einer neuen Service–Version)
 - ▶ Zur Inbetriebnahme muss Schnittstelle stabil sein.

Betrieb von mehreren Versionen eines Services

- ▶ innerhalb *derselben* Laufzeitumgebung
⇒ fachlich getriebene Versionierung
- ▶ in *verschiedene* Umgebungen, z.B. in Entwicklung und Produktion
⇒ Konfigurationsmanagement

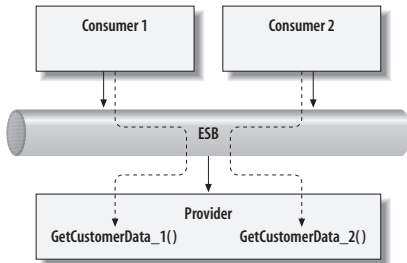
Unterschiedliche Ansätze zur Versionierung

1. Versionierung verstecken

- ▶ Für den Nutzer soll sich bei einer neuen Version keine Änderung ergeben.
- ▶ Rückwärtskompatible Änderungen notwendig
- ▶ Viele verschiedene Versionen im Einsatz
- ▶ Problem: Erkennen der Versionsabhängigkeiten
- ▶ Problem: Verschlanken von Services

2. Versionierung explizit sichtbar

- ▶ Für jede neue Version gibt es einen neuen Service
- ▶ Rückwärtskompatibilität durch Nutzen der *alten* Services
- ▶ Viele verschiedene Versionen im Einsatz
- ▶ Problem: Abschaffen von Services



- ▶ Jede fachliche Änderung führt zu einem neuen Service
- ▶ Namenskonvention, um Versionen zu unterscheiden
- ▶ Service ist bereits in Produktion und nicht mehr in der Entwicklung/Test/Integration
Änderung wird sonst *im Kleinen* mit den Nutzern durchgeführt
- ▶ Service wird bereits benutzt, d.h. die fachliche Änderung hat Auswirkungen auf Nutzer
- ▶ Fehlerbehebungen führen nicht zu neuen Services

Unterstützung der Infrastruktur für

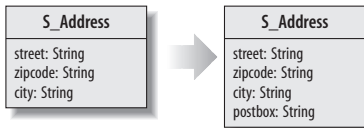
- ▶ transparente Erweiterung von Daten z.B. Hinzufügen eines Attributs
- ▶ Weiterleiten von Aufrufen alter Services auf neue Implementierungen (eine Art Proxy)
- ▶ den Aufruf unterschiedlicher Service–Versionen unter gleichem Namen
 - ▶ Konfiguration der Abbildung, welcher Nutzer welchen Service benutzt

Beispiel (Adress-Daten)

```
String street  
String zipcode  
String city
```

Erweitern durch:

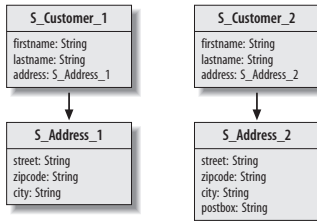
```
String postbox
```



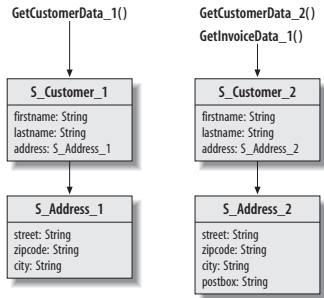
- ▶ Typisierte Schnittstelle mit unterschiedlichen Datentypen
- ▶ Typisierte Schnittstelle mit dem gleichen Datentyp
- ▶ Generische Schnittstelle

Verschiedene Datentypen für verschiedene Versionen

- ▶ Änderungen eines Datentyps führt technisch zu neuem Datentyp
- ▶ Abhängigkeiten zwischen Typen: Änderungen pflanzen sich fort.



- ▶ Verschiedene Services werden möglicherweise verschiedene Versionen der Datentypen verwenden.
- ▶ Problem: Zuweisung und Vergleich nicht ohne weiteres möglich
- ▶ Anbieter verwaltet verschiedene Versionen der Daten
- ▶ Nutzer muss für gleiche fachliche Daten verschiedene Datentypen verwenden. Auch eventuell alte und neue Datentypen.



- 2 Versionen des `GetCustomerData`-Services
- neuer Service `GetInvoiceData` verwendet neueste Version der Kundendaten.

Graphik: *SOA in Practice*, N. Josuttis

Service-
Lebenszyklus
und
Versionierung

Lebenszyklus

Versionierung

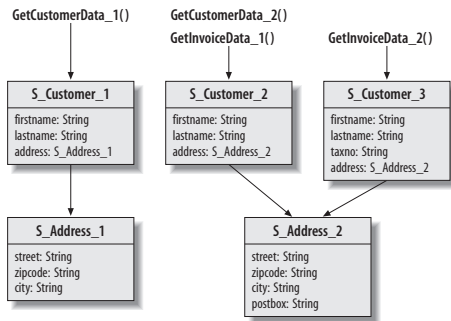
Fachlich getrieben

Datentypen

Konfigurationsmanagement

Ausblick

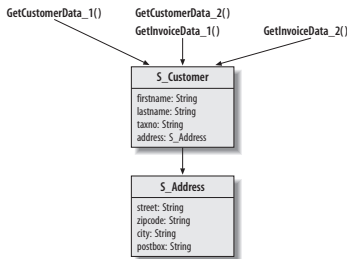
- Neue Anforderung: `GetInvoiceData` soll Steuernummer mitliefern: `GetInvoiceData_2` liefert `S_Customer_3`-Daten



- Neuer Nutzer benötigt Kundendaten und Rechnungsdaten
- Benutzung in jeweils neuester Version
- Muss mit `S_Customer_2`- und `S_Customer_3`-Daten umgehen
- Zuweisung, Vergleich — Mapping im Nutzer nötig
- Aus Nutzersicht kritikwürdig

Der gleiche Typ für verschiedene Datentypversionen

- ▶ Alle fachlichen Versionen der Datentypen verwenden die selben technischen Datentypen.
- ▶ Enthalten jeweils alle Attribute



- ▶ Welche Attribute sind bei welchen Aufruf gültig?
⇒ Dokumentation notwendig (potentiell kompliziert)
- ▶ Technische Datentypen wachsen
- ▶ keine Binärkompatibilität
⇒ Bibliotheksversionen müssen passen
- ▶ Validierung der Daten muss richtige Attribute berücksichtigen.

Service–
Lebenszyklus
und
Versionierung

Lebenszyklus

Versionierung

Fachlich getrieben

Datentypen

Konfigurationsmanagement

Ausblick

- ▶ Ein generischer Datentyp zur Darstellung aller fachlichen Daten
- ▶ z.B. Key-Value-Paare oder XML
- ▶ dynamische Verarbeitung der Daten zur Laufzeit

Beispiel (statische Datentypen)

```
S_Customer_1 custData;  
S_Address_1 address;  
String street;  
input.setCustomerID(id);  
custData = serviceAPI.getCustomerData_1(input);  
address = custData.getAddress();  
street = address.getStreet();
```

Beispiel (generische Datentypen)

```
Data custData, address;  
String street;  
input.setValue("customerID", id);  
custData = serviceAPI.getCustomerData_1(input);  
address = custData.getValue("address");  
street = address.getValueAsString("street");
```

Konfigurationsmanagement-getriebene Versionierung von Services

Service– Lebenszyklus und Versionierung

Lebenszyklus

Versionierung

Fachlich getrieben

Datentypen

Konfigurationsmanagement

Ausblick

- ▶ Verschiedene Versionen eines Services befinden sich in unterschiedlichen Phasen des Service–Lebenszyklus
Produktion — Integration — Test — Entwicklung
- ▶ Anforderung: Bereitstellung zusammengehöriger Artefakte
- ▶ Verwaltung bei Dateien durch Werkzeug zur Versionskontrolle (CVS, ClearCase, Subversion, ...)
- ▶ Spezielle Mechanismen notwendig, wenn Artefakte keine Dateien sind:
 - ▶ Versionierung in der Datenbank
 - ▶ organisatorische Regelungen

Service–Lebenszyklus

Versionierung

Fachlich getriebene Versionierung
Versionierung von Datentypen
Konfigurationsmanagement

Ausblick

- Fragen?

**Service–
Lebenszyklus
und
Versionierung**

Lebenszyklus

Versionierung

Fachlich getrieben

Datentypen

Konfigurationsmanagement

Ausblick

Behandelte SOA–Themen

- ▶ Services
- ▶ Lose Kopplung
- ▶ Enterprise–Service–Bus
- ▶ Service–Klassifizierung
- ▶ Geschäftsprozessmanagement
- ▶ BPEL
- ▶ Organisatorische Aspekte
- ▶ Muster des Nachrichtenaustauschs
- ▶ Service–Lebenszyklus
- ▶ Versionierung

Service– Lebenszyklus und Versionierung

Lebenszyklus

Versionierung

Fachlich getrieben

Datentypen

Konfigurationsmanagement

Ausblick

Offene SOA–Themen

- ▶ Performance
- ▶ Sicherheit
- ▶ Web-Services
- ▶ konkrete Service–Entwicklung
- ▶ konkrete Infrastrukturen