

# **Testgetriebene Entwicklung und kontinuierliche Integration mit der SAP Mobile Platform**

MAXIMILIAN AZIMI (29287)  
JAN-HENRICH MATTFELD (29866)

## **BACHELORARBEIT**

eingereicht im  
Fachhochschul-Bachelorstudiengang

Wirtschaftsinformatik

in Bremerhaven

am 23. März 2015

Erstprüfer: mgr inż. Alfred Schmidt

Zweitprüferin: Prof. Dr. Karin Vosseberg

# Erklärung

Wir versichern, dass wir die Arbeit selbstständig verfasst und nur die angegebenen Quellen und Hilfsmittel benutzt haben. Die einzeln zu verantwortenden Teile haben wir im Inhaltsverzeichnis entsprechend gekennzeichnet.

Bremerhaven, am 23. März 2015

Maximilian Azimi, Jan-Henrich Mattfeld

# Inhaltsverzeichnis

<b>Erklärung</b>	<b>i</b>
<b>Abbildungsverzeichnis</b>	<b>iv</b>
<b>Kurzfassung</b>	<b>v</b>
<b>1 Einleitung</b>	<b>1</b>
1.1 Voraussetzungen . . . . .	1
1.2 Problemstellung . . . . .	1
1.3 Zielsetzung . . . . .	2
1.4 Erkenntnisinteresse . . . . .	2
<b>2 Grundlagen</b>	<b>3</b>
2.1 Was ist testgetriebene Entwicklung? / <i>Mattfeld</i> . . . . .	3
2.1.1 Vorteile der Methode . . . . .	4
2.1.2 Wege zum Erfolg . . . . .	5
2.1.3 TDD mit JavaScript . . . . .	7
2.2 SAP Mobile / <i>Mattfeld</i> . . . . .	9
2.2.1 UI5-Interns . . . . .	9
2.2.2 Architekturen . . . . .	10
2.2.3 Open-Source-Initiative als Chance . . . . .	12
2.3 Neues Backend: SAP Gateway / <i>Azimi</i> . . . . .	12
2.3.1 Best Practices . . . . .	13
2.3.2 Deployment-Optionen . . . . .	14
2.3.3 OData und REST . . . . .	15
<b>3 Anforderungsanalyse</b>	<b>18</b>
3.1 App-Spezifikation / <i>Mattfeld</i> . . . . .	18
3.1.1 Anforderungen . . . . .	18
3.1.2 Architektur . . . . .	19
3.2 Testkonzept . . . . .	20
3.2.1 Steuerung und Planung / <i>Mattfeld</i> . . . . .	20
3.2.2 Durchführung . . . . .	23

<b>4</b>	<b>Implementierung</b>	<b>26</b>
4.1	Backend / <i>Azimi</i> . . . . .	26
4.1.1	Werkzeuge . . . . .	26
4.1.2	Funktionsbausteine . . . . .	27
4.1.3	OData-Service . . . . .	30
4.2	CI-Toolchain . . . . .	38
4.2.1	Paketmanager / <i>Mattfeld</i> . . . . .	38
4.2.2	Statische Analyse mit ESLint / <i>Mattfeld</i> . . . . .	40
4.2.3	Testtreiber Karma / <i>Mattfeld</i> . . . . .	41
4.2.4	Hybrid-App per PhoneGap Build / <i>Azimi</i> . . . . .	44
4.2.5	JS Task Runner Grunt / <i>Azimi</i> . . . . .	45
4.2.6	Jenkins / <i>Azimi</i> . . . . .	47
4.2.7	Zusammenfassung / <i>Azimi</i> . . . . .	51
4.3	UI5-App / <i>Mattfeld</i> . . . . .	52
4.3.1	Auswahl der Entwicklungsumgebung . . . . .	52
4.3.2	UI5-App . . . . .	56
4.3.3	Akzeptanztests mit OPA5 . . . . .	58
<b>5</b>	<b>Schlussfolgerungen</b>	<b>60</b>
5.1	Lessons Learned . . . . .	60
5.1.1	Architektur-Überlegungen . . . . .	60
5.1.2	OData-Service generieren oder selbst implementieren? . . . . .	60
5.1.3	Wo ist die aktivste UI5-Community? . . . . .	61
5.1.4	Infrastruktur als Code . . . . .	61
5.1.5	Alternative IDE prüfen . . . . .	62
5.1.6	Testmethoden kombinieren . . . . .	62
5.1.7	Tests gezielt einsetzen – Nicht doppelt testen . . . . .	63
5.1.8	Statische Analyse differenziert betrachten . . . . .	63
5.1.9	Code Coverage ist nicht alles . . . . .	63
5.2	Ausblick . . . . .	64
5.2.1	<i>Echte</i> Unit-Tests . . . . .	64
5.2.2	Native Funktionen testen . . . . .	64
5.2.3	Tiefere SMP-Integration . . . . .	65
5.2.4	Fiori Deployment . . . . .	65
5.2.5	Continuous Delivery mit Afaria . . . . .	66
5.2.6	eCATT-Integration . . . . .	66
5.3	Bewertung und Herausforderungen . . . . .	66
	<b>Quellenverzeichnis</b>	<b>68</b>
	Literatur . . . . .	68
	Online-Quellen . . . . .	69

# Abbildungsverzeichnis

2.1	TDD-Entwicklungskreislauf . . . . .	3
2.2	Angepasste CI-Toolchain . . . . .	8
2.3	UI5-Eigenentwicklung im Fiori Launchpad . . . . .	11
2.4	SAP-Gateway-Integrationsmodell . . . . .	13
3.1	App-Architektur . . . . .	19
4.1	OData Modeler . . . . .	27
4.2	Funktionsbaustein – Testdatenverzeichnis . . . . .	30
4.3	Funktionsbaustein – Regressionstest . . . . .	30
4.4	Import-RFC-Funktionsbaustein . . . . .	32
4.5	RFC-Mapping . . . . .	33
4.6	RFC-Mapping – Zuordnung der Attribute . . . . .	33
4.7	Entitätstypen – Assoziationen . . . . .	36
4.8	Entitätstypen – Navigationseigenschaften . . . . .	36
4.9	OData-Testfalldatenbank . . . . .	37
4.10	OData-Mehrfachtest . . . . .	37
4.11	PhoneGap-Build-Webinterface . . . . .	44
4.12	Build-Auslöser . . . . .	47
4.13	Integration von Paketmanagern und Grunt . . . . .	48
4.14	Checkstyle-Warnungen – Trend-Graph . . . . .	48
4.15	Checkstyle-Warnungen – Hohe Priorität . . . . .	49
4.16	Zeilenabdeckung im Paket <i>app</i> . . . . .	49
4.17	Zeilenabdeckung im Quellcode . . . . .	50
4.19	Build-Status . . . . .	50
4.18	Plato-Bericht zu <i>Master3.controller.js</i> . . . . .	51
4.20	WebStorm mit TDD-Integration . . . . .	55
4.21	UI5-App – Hauptbildschirm . . . . .	57

# Kurzfassung

Mit aktuellen Tools und Frameworks wie SAPUI5, Gateway und Mobile Platform bietet die SAP neue Wege zur Entwicklung von geräteübergreifenden Anwendungen. Bestehende Individualsoftware auf ABAP-Basis wollen wir mit deren Hilfe mobil nutzbar machen.

Gleichzeitig automatisieren wir den testgetriebenen Entwicklungs- und Auslieferungsprozess. Wir evaluieren entsprechende Open Source Tools und geben Hinweise auf aktuelle Best Practices.

# Kapitel 1

## Einleitung

### 1.1 Voraussetzungen

Vorausgegangen sind dieser Arbeit u. a. die Zertifizierung als ISTQB Certified Tester Foundation Level und mehrere Projekte im SAP-Mobilbereich.

Bisher waren Apps mit Zugriff auf SAP-Systeme vollständige Eigenentwicklungen auf Basis externer Frameworks – SAP bietet mit SAPUI5, Gateway und Mobile Platform neue Möglichkeiten zur Entwicklung von geräteübergreifenden Anwendungen aus einer Hand. Wir wollen sie nutzen, um bestehende Individualsoftware auf ABAP-Basis mobil nutzbar zu machen.

Gleichzeitig automatisieren wir den Entwicklungs- und Auslieferungsprozess. Wir evaluieren entsprechende Open Source Tools und geben Hinweise auf Best Practices. Die SAP-Grundlagen erweitern wir um aktuelle Vorgehensmodelle und Entwicklungsmethoden im Sinne der testgetriebenen Entwicklung.

### 1.2 Problemstellung

Viele Projekte der abat AG arbeiten agil z. B. per Scrum. Zur Organisation ist ein umfangreiches Projektmanagement-Tool als ABAP-Eigenentwicklung vorhanden.

Dieses enthält allerdings weder ein Scrum-Board noch eine mobile Ansicht – schneller Zugriff auf wichtige Funktionen ist unterwegs unmöglich. Die Aufgaben können nur am PC mit Intranet-Zugang bearbeitet werden.

Der aktuelle Workflow: Ausdrucken der einzelnen Aufgaben, anpinnen, manuell verschieben und parallel per Scrum-Transaktion in das SAP-System übertragen. Dies gilt es mit aktuellen Technologien zu vereinfachen.

### 1.3 Zielsetzung

Im Rahmen der Bachelorarbeit wird eine geräteübergreifende App entwickelt, die das Scrum-Board visualisiert und den Zugriff auf Projektdaten schneller und einfacher gestaltet.

Während der testgetriebenen Entwicklung sollen aktuelle Technologien und Tools zum Einsatz kommen.

In Zukunft sollen die Projektaufgaben mit Zusatzinformationen auf einem Smartphone oder Tablet angezeigt und bearbeitet werden. Ein typischer Vorgang in dieser App wird beispielsweise die Statusänderung von Aufgaben – Diese kann per Drag and Drop deutlich schneller erledigt werden.

Der bedeutendste Vorteil ergibt sich aus der ständigen Verfügbarkeit des Projektstatus: Das Scrum-Board muss nicht mehr physisch vorhanden sein, ein Blick in die App genügt. Der umständliche Zugriff über die alte, sehr umfangreiche SAP-Transaktion ist nur noch selten notwendig.

### 1.4 Erkenntnisinteresse

Besonders hervorzuheben ist die Kombination der verschiedenen Aspekte und Vorgehen:

1. Entwicklung einer aktuellen SAPUI5-App für ein bereits vorhandenes Altsystem auf ABAP-Basis.
2. Die Integration des neuen NetWeaver Gateways und der entsprechenden OData-Services.
3. Nutzung des Frameworks für Logon- und Offline-Funktionen.
4. Erstellung von Testfällen anhand der Spezifikation.
5. Zuverlässigkeit vorhandener Features nach Updates durch automatische Regressionstests.
6. Aufbau der Open-Source-CI-Toolchain für eine SAPUI5-Entwicklung.
7. Kontinuierliche Bereitstellung neuer App-Versionen für verschiedene Gerätetypen.

Für alle folgenden Projekte werden diese Aspekte essentiell sein: Es gilt eine entsprechende Toolchain zu erproben und zu etablieren, um Softwarequalität und Erfüllung der Spezifikation nachhaltig zu gewährleisten.



## Kapitel 2

# Grundlagen

### 2.1 Was ist testgetriebene Entwicklung?

Beispielhaft zeigt Abbildung 2.1 den Kern der Methode – Die Entwicklung erfolgt in 3 Schritten:

**Rote Phase:** Eine noch nicht implementierte Funktion wird durch einen Test geprüft. Dieser schlägt fehl – Ist also *rot*.

**Grüne Phase:** Die Funktion wird implementiert, der Testlauf ist erfolgreich und der Teststatus entsprechend *grün*.

**Refaktorisierung:** Der Quellcode wird verbessert, Redundanzen entfernt. Diese Schritte wiederholen sich in einer Endlosschleife während der gesamten Entwicklungszeit. Im Unterschied zu klassischen Herangehensweisen werden zuerst Tests geschrieben und erst danach Funktionalitäten implementiert.

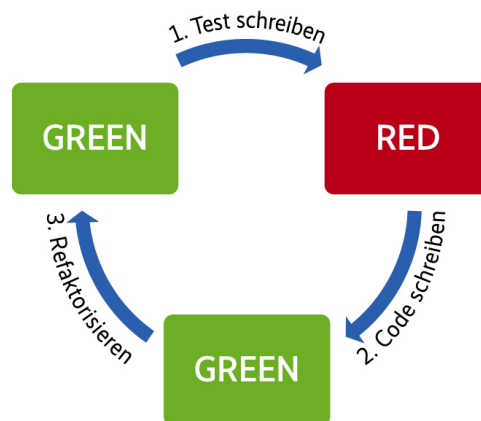


Abbildung 2.1: TDD-Entwicklungskreislauf aus [18].

Referenz	Zeitraum	% Wartung
Pigoski (1997)	1985-1989	75 %
Frazer (1992)	1990	80 %
Pigoski (1997)	1990s	90 %

**Tabelle 2.1:** Anteil der Wartungskosten in Softwareprojekten nach [7, S. 229].

### 2.1.1 Vorteile der Methode

Laut Beschreibung scheint Test Driven Development (TDD) gleichbedeutend mit höherem Aufwand durch den zusätzlichen oder zumindest umfangreicheren Testumfang. Wo ist der Mehrwert?

Zum Beispiel nach Tabelle 2.1 liegt der Anteil an Wartungskosten innerhalb eines Softwareprojektes bei mindestens 80 % – Tendenz weiter steigend. Gemeint sind Aufwände für Erweiterungen und Fehlerbehebungen. Die Hauptaufgabe der Methode TDD ist folglich die langfristige Reduzierung des Wartungsaufwands und damit eine Kosteneinsparung.

#### Keine Schnellstarts

Klassische Entwicklungsmethoden verleiten zu Schnellstarts – Erst programmieren, dann die Anforderungen verfeinern und nachbessern. Der Zwang erst Tests zu schreiben, führt fast zwangsläufig zu einer intensiveren Analysephase, da die zu lösenden Probleme erst verstanden und zerlegt werden müssen. Die Entwicklung wird auf die richtigen Funktionalitäten fokussiert. Daraus ergeben sich weitere Vorteile in Bezug auf Code-Umfang und -Qualität.

#### Weniger Code – Besserer Code

Durch die Zerlegung in testbare Teilprobleme wird von Anfang an eine modulare Software-Architektur aufgebaut. Die Abhängigkeiten sind klar definiert, die Anwendung kann leicht erweitert oder korrigiert werden.

Im Optimalfall entsteht nur Code, der auch getestet wird. Jede Funktion muss ein vorher festgelegtes Teilproblem lösen und einen Test erfüllen. Dieses Vorgehen verhindert die Entstehung von unnötigem Code und ermöglicht die Konzentration auf das Wesentliche. Die Refactoring-Phase sorgt für die Eliminierung von Code-Duplikaten. Jede Aufgabe wird nur einmal gelöst.

#### Jeder Fehler nur einmal

Die stetige Überarbeitung sorgt für geringe *technische Schulden* im Code [12, S. 13]. Weiterführend ändert sich auch der Umgang mit Bugs: Für jeden festgestellten Fehler wird ein eigener Testfall geschrieben. Erst danach beginnt

die Korrektur im Code. Hierdurch werden automatische Regressionstests möglich – Jeder Fehler wird nur einmal begangen.

All dies führt zu einer wartungsfreundlicheren Applikation, die während ihres Lebenszyklus, trotz höherem Initialaufwands, im Vergleich Kosten einsparen kann. Was ist nötig um TDD erfolgreich einzusetzen?

### 2.1.2 Wege zum Erfolg

TDD erfordert viel Umgewöhnung – Einen Test für nicht-existierende Funktionalitäten zu schreiben erscheint ungewöhnlich. Um die Methode trotzdem zum Erfolg zu führen helfen:

- Hochintegrierte Entwicklungsumgebungen
- Automatisierte Tests
- Continuous Integration (CI)
- Transparenz
- Übung und Schulungen

Im Folgenden die wichtigsten Richtlinien, die wir für dieses Projekt aus TDD-Best-Practices [9, S. 99] übernommen haben.

#### Unterstützende Entwicklungsumgebung

Auch wenn einigen Programmierern ein einfacher Texteditor und die Kommandozeile genügen (zum Beispiel [8]) – Wichtig sind möglichst wenig Kontextwechsel während der Entwicklung, und damit eine hochintegrierte Entwicklungsumgebung. Das Werkzeug soll in den Hintergrund rücken und TDD aktiv unterstützen.

Konkret benötigen wir Syntax-Hervorhebung der genutzten Sprache (JavaScript), die Integration von statischen Code-Analysen, Unit-Tests, Test-Treibern, Build-Systemen, Versionsverwaltungen, Verwaltung von Abhängigkeiten und einigen weiteren Funktionen, die wir in Unterabschnitt 4.3.1 erläutern.

#### Testfallanforderungen

Die Codequalität eines Tests muss grundsätzlich den gleichen Ansprüchen genügen, wie die des Hauptcodes. Auch die Tests werden schließlich während des Softwarelebenszykluses immer wieder gewartet und erweitert. Lesbarkeit und Modularität sind entscheidende Erfolgsfaktoren. Im Optimalfall sind die Tests sprechend und selbsterklärend formuliert.

Eine Folge der Modularität ist Unabhängigkeit. Ein Testfall sollte nicht vom Erfolg eines anderen Testfalls abhängen, sondern allein lauffähig sein. Während eines Testlaufs würden sonst nach einem Fehlschlag auch alle weiteren Tests abbrechen. Ohne Testfallunabhängigkeit ist keine Prüfung einzelner Funktionen möglich.

Dies führt zu einem weiteren Problem, dem Ressourcenverbrauch. Um dem Entwickler zeitnah Rückmeldung zu geben, müssen Tests so schnell wie möglich abgearbeitet werden.

Wichtigste Regel: Nur ein Testfall pro Test. So kann einer Fehlerwirkung direkt eine Fehlerursache zugeordnet werden. Die Fehleranalyse bleibt kurz und effizient.

### **Automatisiere und sprich darüber**

Über eine nachgelagerte Test-Abteilung ist die direkte Rückmeldung an den Entwickler nicht möglich. Stattdessen sorgt ein CI-Server für die automatische Ausführung von statischen und dynamischen Tests. Nach dem Einchecken in das Haupt-Repository sind alle Code-Bestandteile verschiedener Entwickler auf der gleichen Grundlage getestet. Veränderungen können grundsätzlich von jedem Entwickler an jeder Stelle nachvollziehbar durchgeführt werden.

Metriken bilden eine wichtige Grundlage für die Entwicklung im Team. Sie beschreiben z. B. die Abdeckung des Codes durch Unit-Tests, Komplexität/Wartbarkeit und andere Ergebnisse der statischen Code-Analyse. Die, hoffentlich positive, Entwicklung der Qualität im Projekt ist so öffentlich verfügbar.

Als Zielvorgaben sind Metriken jedoch ungeeignet. Dies würde zu engen Anpassungen an die Applikation führen, um die geforderte Abdeckung zu erreichen – Die Tests wären schlecht wartbar und gefährden sogar den Erfolg der Methode TDD.

### **Die Mischung macht's**

Testautomatisierung allein löst nicht alle Probleme: Nicht alles ist test- oder automatisierbar. Eine Teststrategie ist unabdingbar [10, S. 33]. Um die richtigen Tests zu schreiben, bieten sich Referenzen an. Neben der Software-spezifikation existieren diverse Testorakel für bestimmte Plattformen wie iOS und Android. Wichtig ist auch Testerfahrung, z. B. im Umgang mit langen Zeichenketten und Sonderzeichen.

Diese Erfahrungen unterstützen besonders im Zusammenhang mit explorativem Testen. Ergänzend unterstützen weitere Techniken wie Acceptance Test Driven Development (ATDD) und Behavior Driven Development (BDD). Kunde und Entwickler entwickeln gemeinsam Akzeptanzkriterien und entsprechende Beispiele. Die Sprache orientiert sich an der Geschäftslogik. Implementierungsdetails wie das Aussehen der grafischen Oberfläche finden hier noch keine Beachtung. Auf Basis der Beispiele werden Tests entwickelt, die auch vom Kunden gelesen werden können. TDD stellt sicher, dass der Code richtig geschrieben wurde, während ATDD dafür sorgt, dass es auch der richtige Code ist.

Letztlich ist der Projekterfolg mit TDD abhängig vom Rückhalt bei

Kunde und Management. Der Zwiespalt zwischen Zeit, Qualität und Kosten ist schwierig zu vermitteln. Zumal TDD und davon abgeleitete Techniken ihre Stärken nicht bei Prototypen, sondern hauptsächlich durch Verringerung des Wartungsaufwands in längeren Projekten ausspielen.

### 2.1.3 TDD mit JavaScript

Grundsätzlich ist die testgetriebene Herangehensweise unabhängig von Programmiersprachen nutzbar. So existieren für alle bekannten Sprachen Unit-Test-Frameworks: JUnit für Java, PHPUnit für PHP usw.

#### Test-Tool-Vielfalt

JavaScript ist anders: Es gibt kein Standard-Test-Framework. Verbreitet sind Eigenentwicklungen der größeren JavaScript-Frameworks wie QUnit für jQuery, oder unabhängige Bibliotheken wie Jasmine.

Unterscheiden lassen sich zumindest client- und serverseitige Test-Frameworks. Die beiden bereits genannten erfordern eine HTML-Infrastruktur, in der sie die Tests im Browser ausführen. Interessant für die Nutzung im CI-System sind Bibliotheken wie JSTestDriver oder Karma, die über eine eigene Serverkomponente mehrere Browser gleichzeitig ansprechen und automatisiert ganze Test-Suiten ausführen.

Ein ähnliches Bild gibt es bei der Vielfalt der JavaScript-Tools zur statischen Code-Analyse. Zur Auswahl stehen mindestens JSLint, JSHint und ESLint. Diese unterscheiden sich lediglich in den Einstellmöglichkeiten und reichen von strengen, nach Douglas Crockford (JSLint), bis zu komplett individuellen Regeln (ESLint).

Für einige Tests ist es nötig, nicht vorhandene Komponenten zu simulieren. Ein Beispiel könnte das, zu Entwicklungsbeginn nicht vorhandene, OData-Backend sein. Man spricht von sogenannten Mock-Objekten, die entsprechende Methodenaufrufe mit Testdaten beantworten. Für einen tieferen Einblick in Methoden sorgt ein Spy: Er protokolliert generell Methodenaufrufe, Parameter und Rückgabewerte – In JavaScript oft Callbacks. Mit Sinon.JS existiert ein Standard-Framework für diese Funktionalitäten.

Spezielle Herausforderungen in JavaScript sind die Abhängigkeit vom Document Object Model (DOM) und Beachtung von asynchronen Zugriffen im Rahmen von Asynchronous JavaScript and XML (AJAX). Beide lassen sich lösen: HTML-Fixtures (Templates) und Promises vereinfachen den Umgang. Letztere sind Bestandteile aller größeren Frameworks, z. B. in Form des jQuery-Deferred-Objekts. Es fungiert als Proxy für Callbacks und verwaltet die Abhängigkeiten asynchroner Aufrufe.

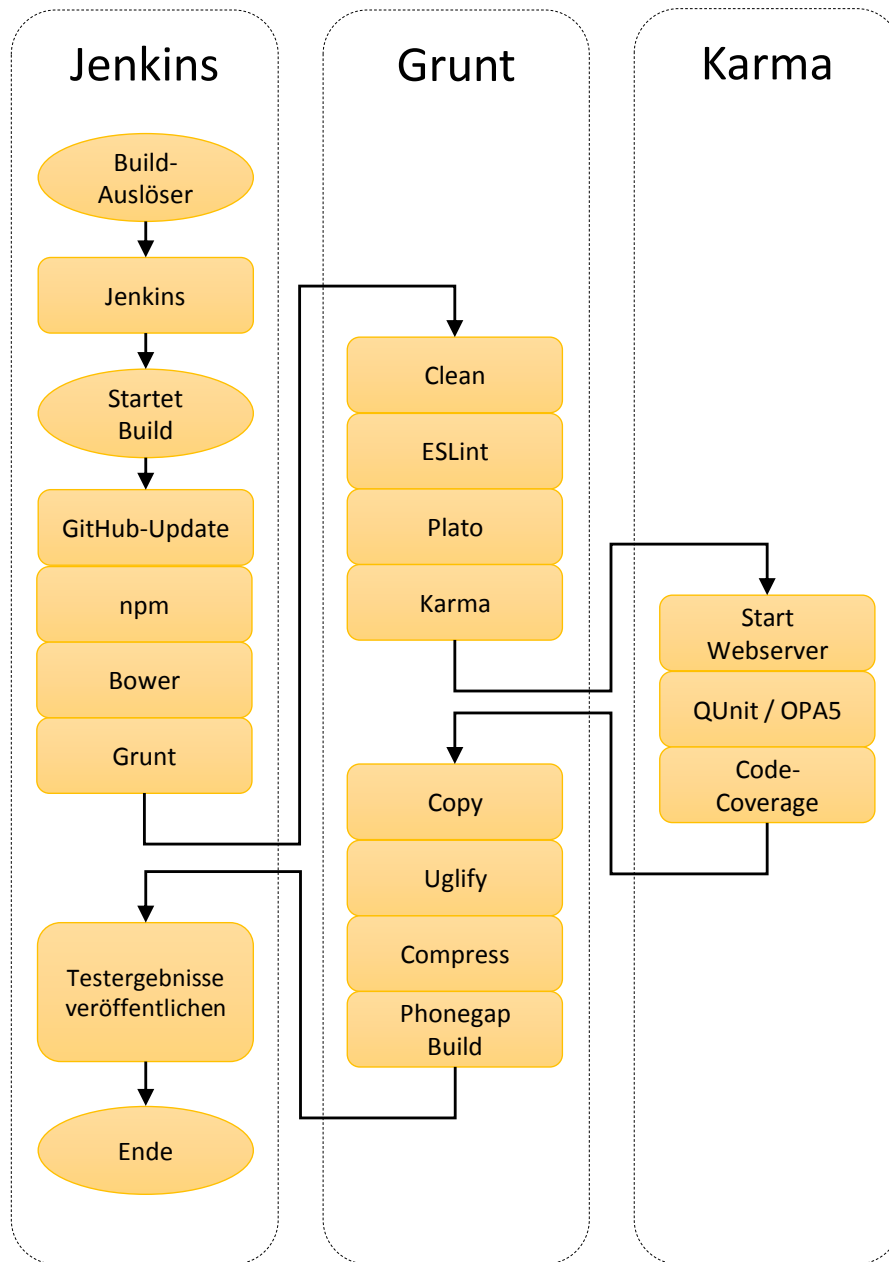


Abbildung 2.2: Angepasste CI-Toolchain.

### Projektverwaltung

Allein die Anzahl der bisher erwähnten JavaScript-Test-Frameworks ist hoch. Für einen kompletten Build-Vorgang fehlen zurzeit sogar noch Tools. Wie in Abbildung 2.2 gezeigt, muss der Code nach dem Test noch bereinigt, kompri-

miert und veröffentlicht werden. Diese Aufgaben übernimmt ein JavaScript TaskRunner wie Grunt oder Gulp mit wiederum eigenen Plugins.

Zwischen den genannten Tools und Frameworks bestehen Abhängigkeiten, die ein Paketmanager auflösen sollte [3]. Im Fall der Entwicklungs- und Testtools übernimmt der node.js-Paketmanager npm diese Aufgabe. Für Frontend-Frameworks wie jQuery oder SAPUI5 steht Bower zur Verfügung.

### Viele kleine Helfer

Die Übersicht zu TDD mit JavaScript zeigt: Es gibt eine große Vielfalt an Frameworks. Diese sind oft klein und erfüllen nur spezielle Aufgaben. Die Mehrheit wird in OpenSource-Communities entwickelt und beeinflusst sich gegenseitig stark. Das Entwicklungstempo ist hoch, und so sind bestehende Best Practices innerhalb weniger Monate oder Jahre hinfällig.

Die Herausforderung während der Implementierung wird in der Auswahl der geeigneten Tools und Frameworks liegen: Welche Teile können aus SAPUI5 übernommen werden, was muss durch externe Module ergänzt werden? Ist die große Modularität des Java-Script-Workflows eine Stärke aus der ein Standard entwickelt werden kann?

## 2.2 SAP Mobile

Das UI5 Development Toolkit for HTML5 (SAPUI5) steht im Mittelpunkt der aktuellen SAP-Mobile-Strategie. Als geräteübergreifendes HTML5-Framework soll es schrittweise die SAP GUI und mobile Eigenentwicklungen ablösen. Viele Standardanwendungsfälle werden bereits von den mitgelieferten Fiori Apps auf UI5-Basis abgedeckt. Die ebenfalls neuen Screen Personas sind dagegen nur eine Übergangslösung, um vorhandene Transaktionen vereinfacht darzustellen.

Verfügbar ist UI5 als proprietäres SAPUI5 für SAP-Kunden oder als quelloffenes OpenUI5. Da der Kern beider Versionen identisch ist, sprechen wir im Folgenden nur noch von UI5.

Laut Gartner ist die zugehörige SAP Mobile Platform (SMP) schon jetzt führende Entwicklungs-Plattform für mobile Applikationen [15]. Auch die Ankündigung der SAP, den Mainstream Support der SMP 3.0 bis 2020 zu verlängern [6], unterstreicht den Zukunftscharakter dieser Technologie. Bisherige Lessons Learned aus UI5-Projekten zeigen vor allem Potenzial in den Bereichen Entwicklungsumgebung, Test-Entwicklung und -Automatisierung [14], die wir daher besonders betrachten.

### 2.2.1 UI5-Intern

Das Framework besteht aus mehreren Bibliotheken: Immer geladen werden *sap.ui.core* und *sap.ui.commons*, das Standard-Bedienelemente beinhaltet.

Für unsere App werden wir außerdem *sap.m* nutzen, das seine UI-Elemente dynamisch an die Bildschirmgröße anpasst. Während diese Bibliotheken auch im quelloffenen OpenUI5 vorhanden sind, haben nur zahlende Kunden Zugriff auf *sap.viz* und *sap.makit* zur Diagrammerstellung.

Die Entwicklung von SAPUI5 begann schon 2011, ist jedoch keine vollständige Neuentwicklung. Integriert werden verschiedene Open-Source-Frameworks wie jQuery, jQuery Mobile, data.js, QUnit und Sinon.JS – Kombiniert ergeben sie ein JavaScript-Framework, dass sich auch für umfangreiche Unternehmensanwendungen eignet und durch jQuery-Plugins oder eigene Steuerelemente erweitert werden kann.

Im Unterschied zu jQuery Mobile als Einzelframework unterstützt es Modularisierung, Internationalisierung, Routing, Testbarkeit und vieles mehr. Es befindet sich damit auf einer Ebene mit anderen Enterprise Frameworks wie Sencha Ext JS inkl. Sencha Touch oder AngularJS, die ähnliche Ansätze verfolgen.

UI5-Apps basieren auf der Model View Controller (MVC)-Architektur. Die Anwendungslogik wird in den Browser verlagert. Der zuständige Controller ist grundsätzlich in JavaScript implementiert. Views werden dynamisch ebenfalls per JavaScript erzeugt oder statisch als XML, HTML oder JSON definiert. Das Model bezieht die App über Representational State Transfer (REST)-Services.

### 2.2.2 Architekturen

Das SAP Gateway liefert entsprechende OData-Services nach dem REST-Prinzip:

- Zustandslos – Operationen sind in sich abgeschlossen
- Übertragung im XML-Format
- Navigationseigenschaften – z. B. Verlinkung vom Kunden zum Projekt
- URL-adressierbare Objekte – z. B. *http://.../Kunde(3)/Projekt(1)*
- Operationsbasiert – z. B. *GET, POST, PUT, DELETE, ...*

Diese Services lassen sich auf Basis verschiedener Quellen implementieren und sind insgesamt leichter handhabbar als ihre Vorgänger SOAP/WSDL. Abschnitt 2.3 zeigt die wichtigsten Möglichkeiten.

Die für mobile Geschäftsanwendungen oft geforderte Offline-Fähigkeit ergibt sich allerdings erst im Zusammenspiel mit der *SAP Mobile Platform 3.0* oder ihrem Cloud-Pendant *HANA Cloud Platform mobile services* auf Basis der ehemaligen Sybase Unwired Platform. Single Sign-on (SSO), Push-Benachrichtigungen und Fernzugriff ohne VPN sind weitere Zusatzfunktionen beider Lösungen.

Daraus folgen mindestens fünf Deployment- und App-Architekturen, die sich vor allem in den nutzbaren Business- und Geräte-Funktionen unterscheiden:



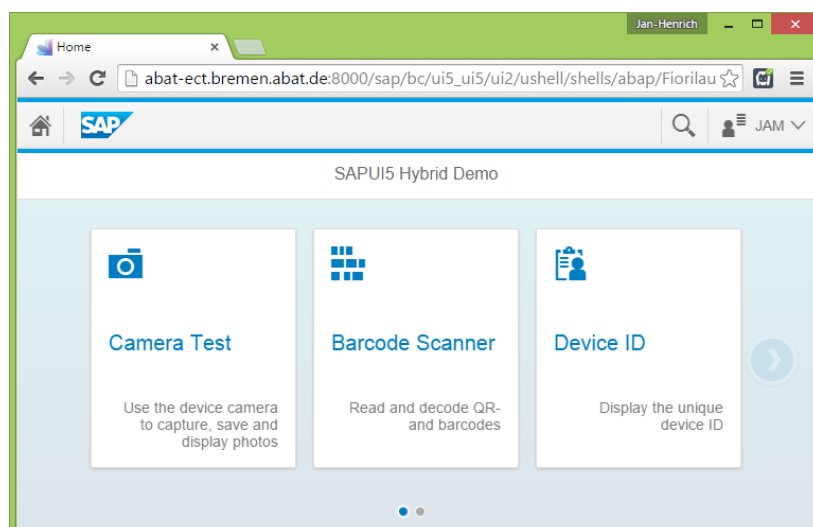
1. Web
2. Fiori-Launchpad-Integration
3. Zugriff über Fiori Client
4. Hybrid-App
5. Angepasster Fiori Client

Methode 1 eignet sich für alle Arten von UI5-Apps. Die Veröffentlichung kann auf jedem beliebigen Webserver erfolgen. Ein SAP-Bezug ist nicht nötig. Allerdings werden keine weiteren Business- oder Geräte-Funktionen unterstützt.

Zur Fiori-Launchpad-Integration ist eine bestimmte App-Struktur und Modularisierung nötig. Die App wird normalerweise über ein ABAP-Repository veröffentlicht. Vorteile sind die SSO-Möglichkeit und ein einheitliches Benutzererlebnis durch die Integration in Fiori wie in Abbildung 2.3.

Der Fiori Client ist für iOS und Android verfügbar. Wird das Launchpad durch ihn aufgerufen, speichert er Zugangsdaten und App-Inhalte im Cache: Die Performance wird verbessert. Weitere Gerätefunktionen sind zur Zeit nicht nutzbar.

UI5-Apps lassen sich beliebig in eigene Hybrid-Apps auf Cordova-Basis integrieren, um native Gerätefunktionen wie Kamera oder GPS aufzurufen. Geschäftsfunktionen und der Zugriff auf die SAP-Infrastruktur gelingt aber nur mit einem angepassten Fiori Client. Der Client-Quellcode ist als Teil des Mobile Platform SDKs verfügbar. Zusammen mit den sogenannten Kapsel Plugins (Cordova) ist das der einzige Weg, um die Offline- und Push-Funktionen der Mobile Platform (SMP) nutzbar zu machen.



**Abbildung 2.3:** UI5-Eigenentwicklung im Fiori Launchpad.

### 2.2.3 Open-Source-Initiative als Chance

Historische Entwicklungen auf Basis der SAP-eigenen Programmiersprache ABAP (Advanced Business Application Programming) lassen sich nur schwer in einem CI-Prozess automatisieren: Die entsprechenden Werkzeuge z. B. zum ABAP-Unit-Test [5] oder zur Testfallerstellung (eCATT) liegen vor, lassen sich aber nur schwer einbinden. Eine weitere Rolle spielt die grundsätzliche ABAP-Entwicklung im System selbst – Eine CI-Interaktion von außen gestaltet sich schwierig.

Deutlich mehr Möglichkeiten ergeben sich bei SAP-Entwicklungen auf Java-Basis: Hier steht die SAP NetWeaver Development Infrastructure (NWDI) zur Verfügung [17]. Alle CI-relevanten Tasks sind vorhanden und lassen sich automatisieren. NWDI ist allerdings proprietär und eignet sich nicht zur Entwicklung mit anderen Plattformen als Java EE.

Eine komplette Kehrtwende ergibt sich nun nach der Einführung von SAPUI5 als neue SAP-Mobilplattform: Sie ist auch ohne SAP-Backend lauffähig und basiert auf dem weit verbreiteten JavaScript-Framework jQuery [1]. Zusätzlich sind große Teile des Quellcodes unter dem Namen OpenUI5 auf GitHub veröffentlicht worden – inklusive Buildkonfiguration und ausführlicher Dokumentation [19]. Sie geben einen guten Einblick in den SAP-internen Workflow.

In der Bachelorarbeit wird untersucht, welche Teile für eigene Projekte übernommen werden können. Wo sind größere Anpassungen und Erweiterungen notwendig? Testbarkeit spielte bei der Entwicklung des Frameworks offenbar eine größere Rolle als früher: Frei verfügbar sind bereits der QUnit-Aufsatz OPA5 (One-Page Acceptance tests for UI5) und ein Mock-Server zur Emulation von OData-Services [2].

#### Gute Voraussetzungen

Folgende Aspekte begünstigen den Aufbau einer SAPUI5-CI-Toolchain mit Jenkins (siehe Angepasste CI-Toolchain):

1. Bewährte Basistechnologien
2. Open-Source-Vorstoß
3. Testorientierung
4. Wachsende Community

## 2.3 Neues Backend: SAP Gateway

Das SAP Gateway wurde im Mai 2011 auf den Markt gebracht, um die Reichweite von SAP-Geschäftsanwendungen zu erhöhen. Dazu bietet das Gateway eine offene, REST-basierte Schnittstelle, um einen einfachen Zugriff auf SAP-Systeme zu ermöglichen, siehe Abbildung 2.4. Das offene Protokoll

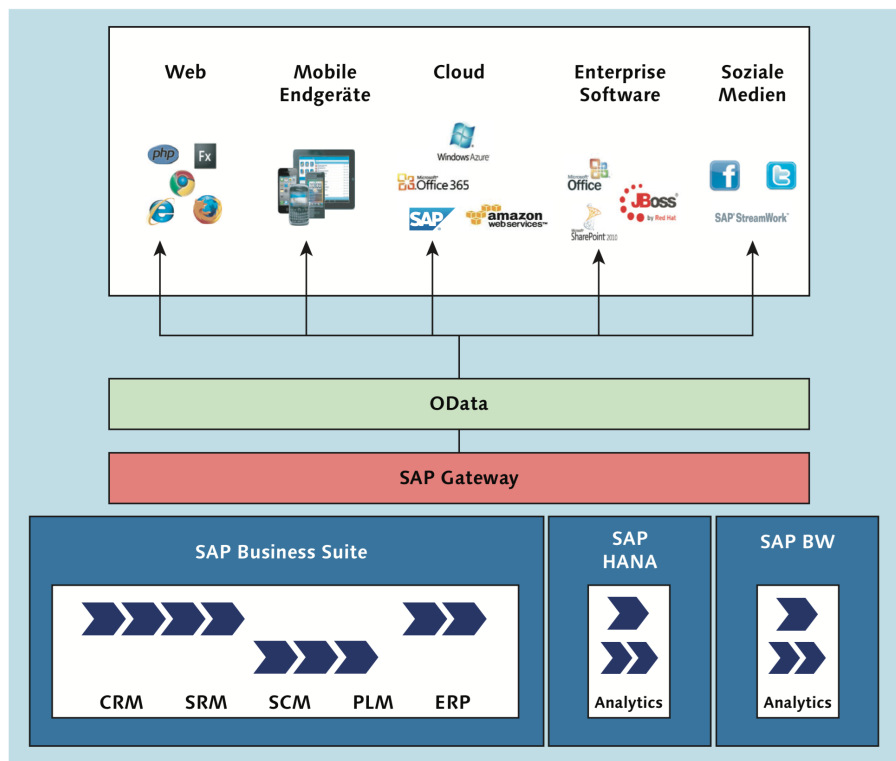


Abbildung 2.4: SAP-Gateway-Integrationsmodell aus [2, S. 45].

OData wird verwendet, da es viel genutzt, bekannt und leicht zu lernen ist. Damit soll die SAP-Welt jedem Entwickler zugänglich sein, der OData versteht, da kein spezielles SAP-Fachwissen mehr nötig ist, um eine Oberfläche für Anwendungen und den Zugriff auf SAP-Daten zu entwickeln.

Durch die Bereitstellung der OData-Schnittstelle müssen der Backend-Entwickler und der Frontend-Entwickler nicht wissen, was der jeweils andere im Detail programmiert. Eine klare Definition der benötigten Schnittstellen ist allerdings zwingend notwendig und erleichtert die gesamte Entwicklung [2, S. 31-45].

### 2.3.1 Best Practices

Um möglichst hochwertige Anwendungen mit Hilfe von OData-Services zu entwickeln, empfiehlt SAP folgende Best Practices zu beachten [2, S. 561-563]:

- Kommunikation zwischen den Entwicklungsteams zur genauen Spezifikation der Schnittstellen.
- Frühzeitiger Entwicklungsbeginn des OData-Services, damit dem Frontend-Entwicklerteam möglichst früh ein konsumierbarer Service zur Verfügung steht.

- Gruppierung der Services in Module, sodass die Anwendung auf bereits vollständig entwickelte Teil-Services zugreifen kann.
- Gründliches Testen der vollständig entwickelten OData-Services mit Hilfe von Test-Tools, sowie zusätzliche Tests mit der entwickelten Anwendung.
- Frühzeitiges Testen in der Produktivumgebung.
- Nutzen der Query-Option `sap-ds-debug=true` im Browser.

### 2.3.2 Deployment-Optionen

Für das Deployment des SAP Gateways stehen verschiedene Optionen zur Verfügung. Diese werden nachfolgend beschrieben.

#### Eingebettetes Deployment

Bei dieser Variante des Deployments werden alle Komponenten des SAP Gateways auf dem SAP-Backend-System installiert. Ab SAP NetWeaver 7.40 ist das Gateway Bestandteil der Standardinstallation. Hauptvorteil ist die Reduzierung der Laufzeit um einen Remote Call: Der Service wird direkt auf dem Backend registriert. Der Nachteil an dieser Variante ist, dass das SAP Gateway für jedes Backend separat konfiguriert werden muss.

Zudem rät SAP davon ab, SAP-Backend-Systeme mit eingebettetem Gateway als Hub-System für andere Backends zu verwenden. Abhängigkeiten im Aktualisierungsprozess können ein Hub-Update bei bereits aktualisiertem Backend-System verhindern [2, S. 52-54].

#### Hub-Deployment mit Entwicklung auf dem SAP-Backend

Die Gateway-Kernkomponenten werden auf einem zusätzlichen SAP-Gateway-Hub-System installiert. Zusätzlich muss auf dem SAP-Backend-System die Business-Enablement-Provisioning-Komponente (*IW\_BEP*) installiert werden. Bei dieser Variante wird der Service auf dem Backend-System entwickelt und veröffentlicht (deployt). Der Service wird anschließend auf dem Gateway Hub registriert. Beim Hub-Deployment ist es möglich, mehrere SAP-Backends mit dem Gateway zu verbinden. Zudem gibt es hier keinen direkten Zugriff auf das SAP-Backend-System [2, S. 54].

#### Hub-Deployment mit Entwicklung auf dem Hub

Alle Komponenten des SAP Gateways werden auf dem Hub-System installiert. Diese Variante wird beispielsweise bei eingeschränktem Zugriff auf das Backend-System eingesetzt, falls eine Entwicklung auf diesem nicht möglich ist. Zwingend ist diese Variante bei Release-Ständen unter SAP NetWeaver 7.40, dort ist eine Installation von *IW\_BEP* auf dem Backend nicht erlaubt.

Die Entwicklung des Services findet auf dem Hub-System statt. Vorteil: Das SAP-Backend-System wird nicht modifiziert. Allerdings muss bei der Entwicklung auf eine bereits vorhandene Schnittstelle wie z. B. einen Remote Function Call (RFC) oder ein Business Application Programming Interface (BAPI) zurückgegriffen werden, die das Backend bereitstellt [2, S. 55].

### 2.3.3 OData und REST

Das Open Data Protocol (OData) ist ein REST-basiertes Datenzugriffsprotokoll, welches unter der Microsoft Open Specification Promise (OSP) herausgegeben wurde und auf verbreiteten Standards wie Atom Pub, XML und JSON aufbaut [2, S. 69-70]. Um eine REST-konforme Architektur einzuhalten müssen folgende Eigenschaften erfüllt werden [2, S. 65-66]:

1. **Client-Server-Architektur** – Client und Server werden von einer einheitlichen Schnittstelle getrennt.
2. **Zustandslosigkeit** – Es werden keine Daten zu den einzelnen Abfragen gespeichert. Daher muss jede Abfrage alle benötigten Informationen beinhalten.
3. **Pufferbarkeit** – Die Antworten des Servers müssen spezifizieren, wie lange Daten gespeichert werden dürfen, damit keine veralteten oder falschen Daten verwendet werden.
4. **Mehrschichtigkeit** – Der Client weiß nicht, ob er direkt mit dem Backend oder nur mit einem Server in der Mitte verbunden ist.
5. **Einheitliche Schnittstelle** – Die Ressourcen werden nach einem einheitlichen Verfahren identifiziert, wie z. B. Name, Pfad oder Primärschlüssel. Abfrage und Manipulation von Ressourcen geschehen nach einem einheitlichen Verfahren.

Durch OData werden dem Konsumenten definierte Schnittstellen für verschiedene Datenquellen zur Verfügung gestellt und soll so die Zusammenarbeit zwischen verschiedenen Plattformen ermöglichen [2, S. 69-70].

### Datenmodell

Für den OData-Service muss ein Entity Data Model (EDM) definiert werden, welches über das Metadaten-Dokument für Clients zum Abruf bereit steht. Im EDM werden die einzelnen Entitäten, Entitätstypen und ihre Attribute definiert. Des Weiteren werden die Kardinalitäten zwischen den einzelnen Entitäten über Assoziationen festgelegt, sodass über Navigationsattribute zwischen den einzelnen Entitäten navigiert werden kann [2, S. 72-73].

### Servicedokumente

Im Servicedokument (siehe Listing 1) werden alle Entitätsmengen, welchen immer ein Entitätstyp zugrunde liegt, eines Services angezeigt. Zudem wer-

den SAP-spezifische Metainformationen angezeigt, welche unter anderem beschreiben, ob die Entitätsmengen das Hinzufügen, Ändern oder Entfernen von Einträgen oder das erlauben. Erreichbar ist das Servicedokument über den Aufruf des Services ohne zusätzliche Parameter [2, S. 77-81]. Die URI zu einem Servicedokument könnte beispielsweise so aussehen:

*http://host:port/sap/opu/odata/sap/Z\_ZAV\_SCRUM\_SRV/*

```
<?xml version="1.0" encoding="utf-8"?>
<app:service xml:lang="de"
  xml:base="http://host:port/sap/opu/odata/sap/Z_ZAV_SCRUM_SRV/"
  xmlns:app="http://www.w3.org/2007/app"
  xmlns:atom="http://www.w3.org/2005/Atom"
  xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata"
  xmlns:sap="http://www.sap.com/Protocols/SAPData">
  <app:workspace>
    <atom:title type="text">Data</atom:title>
    <app:collection sap:label="Sprints" ...>
    <app:collection sap:label="Aufgaben" ...>
    <app:collection sap:label="Status" ...>
    <app:collection sap:label="Projekt" ...>
    <app:collection sap:label="Kunden" sap:creatable="false"
      sap:updatable="false" sap:deletable="false" sap:pageable="false"
      sap:addressable="false" sap:content-version="1" href="Kunden">
      <atom:title type="text">Kunden</atom:title>
      <sap:member-title>Kunden</sap:member-title>
    </app:collection>
  </app:workspace>
  <atom:link rel="self"
    href="http://host:port/sap/opu/odata/sap/Z_ZAV_SCRUM_SRV/">
  <atom:link rel="latest-version"
    href="http://host:port/sap/opu/odata/sap/Z_ZAV_SCRUM_SRV/">
</app:service>
```

**Listing 1:** Servicedokument Z\_ZAV\_SCRUM\_SRV

Im Service-Metadokument (siehe Listing 2) sind die gesamten Metainformationen, inklusive Datenmodell, Entitätstypen, Entitätsmengen, Assoziationen und Navigationseigenschaften des Services enthalten. Dadurch kennt der Zugreifende alle Informationen über Struktur, Ressourcen und Verknüpfungen [2, S. 81-82]. Aufgerufen wird das Service-Metadokument über den zusätzlichen Parameter *\$metadata* wie im folgendem Beispiel:

*http://host:port/sap/opu/odata/sap/Z\_ZAV\_SCRUM\_SRV/\$metadata*

```

<?xml version="1.0" encoding="utf-8"?>
<edmx:Edmx Version="1.0"
  xmlns:edmx="http://schemas.microsoft.com/ado/2007/06/edmx"
  xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata"
  xmlns:sap="http://www.sap.com/Protocols/SAPData">
  <edmx:DataServices m:DataServiceVersion="2.0">
    <Schema Namespace="Z_ZAV_SCRUM_SRV" xml:lang="de"
      xmlns="http://schemas.microsoft.com/ado/2008/09/edm">
      <EntityType Name="Kunden" sap:content-version="1">
        <Key>
          <PropertyRef Name="Kunde"/>
        </Key>
        <Property Name="Kunde" Type="Edm.String"
          Nullable="false" MaxLength="10" sap:label="Kunde"
          sap:creatable="false" sap:updatable="false" sap:sortable="false"/>
        <Property Name="Bezei" Type="Edm.String"
          Nullable="false" MaxLength="40" sap:label="Bezeichnung"
          sap:creatable="false" sap:updatable="false" sap:sortable="false"/>
        <NavigationProperty Name="Projekte"
          Relationship="Z_ZAV_SCRUM_SRV.Assoc_Kunde_Projekt"
          FromRole="FromRole_Assoc_Kunde_Projekt"
          ToRole="ToRole_Assoc_Kunde_Projekt"/>
      </EntityType>
      <Association Name="Assoc_Projekt_Sprint"
        sap:content-version="1">
        <End Type="Z_ZAV_SCRUM_SRV.Projekt"
          Multiplicity="1" Role="FromRole_Assoc_Projekt_Sprint"/>
        <End Type="Z_ZAV_SCRUM_SRV.Sprints"
          Multiplicity="*" Role="ToRole_Assoc_Projekt_Sprint"/>
        <ReferentialConstraint>
          <Principal Role="FromRole_Assoc_Projekt_Sprint">
            <PropertyRef Name="Kunde"/>
            <PropertyRef Name="Projektid"/>
          </Principal>
          <Dependent Role="ToRole_Assoc_Projekt_Sprint">
            <PropertyRef Name="Kunde"/>
            <PropertyRef Name="Projektid"/>
          </Dependent>
        </ReferentialConstraint>
      </Association>
    </Schema>
  </edmx:DataServices>
</edmx:Edmx>

```

Listing 2: Metadaten Z\_ZAV\_SCRUM\_SRV

## Kapitel 3

# Anforderungsanalyse

### 3.1 App-Spezifikation

Kurzbeschreibung der App:

- Darstellung eines Scrum-Boards mit mehreren Spalten
- Darstellung der Aufgaben in den zugehörigen Spalten
- Möglichkeit für den User, Aufgaben zwischen den Spalten hin- und herzuschieben
- Update des Aufgabenstatus in der Datenbank durch Verschieben

Relevante Backend-Tabellen:

- zav\_kunden: Kundentabelle
- zav\_projekte: Projekte je Kunde
- zav\_sprints: Sprints je Projekt je Kunde
- zav\_aufgaben: Aufgaben je Projekt, je Kunde, teilweise Sprints zugeordnet
- zav\_formcast: Definition von Spalten je Projekt je Kunde; Zuordnung von Aufgabenstatus zu Spalten

#### 3.1.1 Anforderungen

Die Startsequenz der App ist wie folgt:

1. Anmeldung am System
2. Auswahl eines Kunden
3. Auswahl eines (zum Kunden gehörigen) Projektes
4. Auswahl eines (zum Projekt gehörigen) Sprints
5. Darstellung eines Scrum-Boards mit mehreren Spalten
6. Darstellung der zu den Spalten gehörigen Aufgaben als Aufgabenzettel



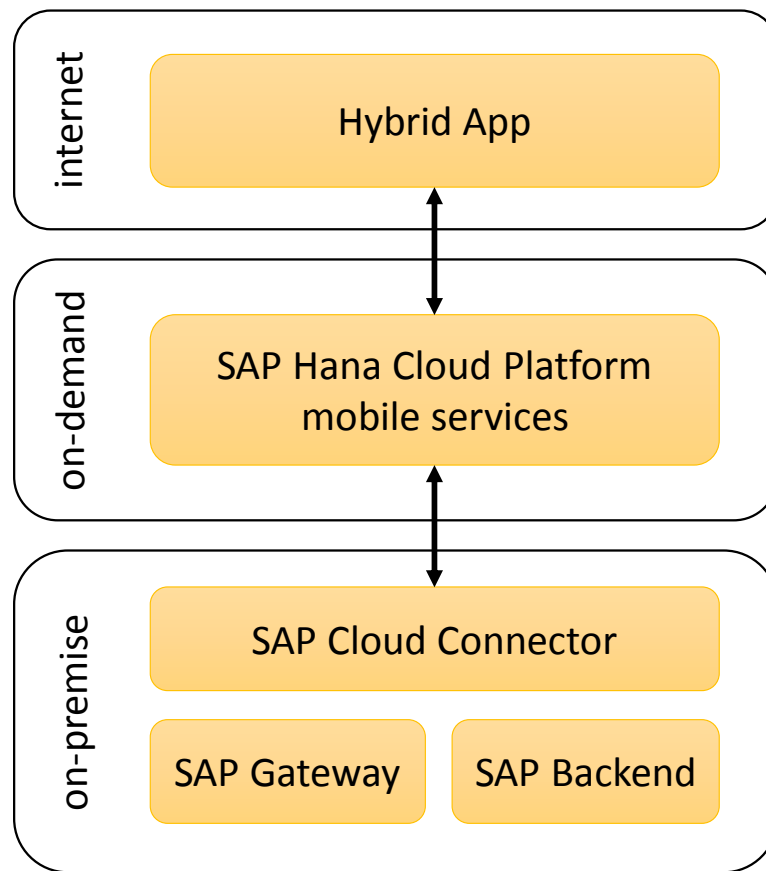


Abbildung 3.1: App-Architektur nach [21].

Die Spalten-Überschriften sind dabei dynamisch, jedoch ergibt sich für jedes Projekt ein jeweils eindeutiger Satz an Spalten. Die Einordnung von Aufgaben in den Spalten ergibt sich aus dem Status der Aufgabe; Die Zuordnung von Status-Konstellationen zu Spalten folgt der Tabelle (ZAV\_FORMCUST).

### 3.1.2 Architektur

Wie im Unterabschnitt 2.2.2 beschrieben, gibt es viele Möglichkeiten eine UI5-App zu betreiben und zu veröffentlichen. Die Entscheidung richtet sich nach den benötigten Geräte- und Geschäftsfunktionen sowie der SAP-Integrationstiefe.

In der ersten Version der Scrum-App sind weder Offline-Funktionen, noch native Gerätezugriffe gefordert. Einzig der Zugriff ohne Intranet-Verbindung erfordert besondere Beachtung. Die empfohlene Lösung ist die Weiterleitung des OData-Services nach außen über die Hana Cloud Platform, siehe Abbildung 3.1. Der SAP Cloud Connector stellt die sichere Verbindung zwischen

SAP-Backend vor Ort und der Cloud her. Vorteil: Das SAP-Backend des Kunden ist nach außen nicht sichtbar, die Firewall-Konfiguration muss nicht angepasst werden. App-Benutzer melden sich direkt an der Cloud-Plattform an, benötigen also keinen Intranet-Zugriff per VPN mehr.

## 3.2 Testkonzept

Das Testkonzept orientiert sich an IEEE 829 [13], angepasst an die kleinere Projektgröße. Es ist als Dokument für alle Aktivitäten während des gesamten Projekts gültig. Darüber hinaus soll es als Vorlage für zukünftige App-Entwicklungen dienen. Ziel ist die Umsetzung des fundamentalen Testprozesses nach Spillner [11]. In anderen Teilen der Arbeit behandelte Testwerkzeuge werden nicht beschrieben. Berücksichtigt werden folgende Dokumente:

- Anforderungsspezifikation des Kunden, siehe Abschnitt 3.1
- UI5 Development Conventions and Guidelines [19], insbesondere:
  - JavaScript Coding Guidelines
  - UI5 Control Development Guidelines
  - Product Standards/Acceptance Criteria
  - Git Guidelines
  - File Names and Encoding
- SAP Gateway Best Practices ([2, S. 561-563])
- Projektvorgehen: Testgetrieben inkl. CI (siehe Kapitel 2)

### Testobjekte

Der Architektur aus Unterabschnitt 3.1.2 folgend, besteht die Anwendung aus zwei Hauptkomponenten. Beide sind Teil des Testkonzepts:

- HTML5-Frontend (OpenUI5 1.26.8)
- SAP-Backend
  - Funktionsbausteine (ECC 6.0, EHP 7, NW 7.4)
  - OData-Service (SAP Gateway NW 7.4)

Nicht einzeln getestet werden die JavaScript-Frameworks wie OpenUI5 und die verwendeten Test-Frameworks. Die Testbasis der entsprechenden Open-Source-Projekte ist mit diversen Unit- und Akzeptanztests sehr umfangreich. Eine projektinterne Test-Wiederholung bringt keinen Mehrwert.

### 3.2.1 Steuerung und Planung

Durch die kleine Teamgröße sind Entwickler gleichzeitig Tester (Test-Manager, Designer, Automatisierer, ...). Sie erarbeiten gemeinsam mit dem Kunden

nach dem Prinzip des BDD allgemein verständliche Anforderungen. Die Beschreibung der Fallbeispiele erfolgt im Schema von automatisierbaren *Wenn-Dann-Sätzen*.

Fehlerkosten verdoppeln sich in jeder Entwicklungsphase. Der Kunde wird daher besonders stark in die Designphase einbezogen, um Kosten durch späte Änderungen zu vermeiden.

### **Risiko und Kosten**

Insgesamt sind Fehlerkosten schwer abzuschätzen. Schadenshöhe und Wahrscheinlichkeit sind durch interne, ergänzende Verwendung der App überschaubar: Die Scrum-Aufgaben werden nur angeschaut oder im Status verändert. Nutzereingaben über den Client müssen allerdings immer als kritisch eingestuft werden. Gültigkeitsprüfungen in der App dienen nur der schnellen Rückmeldung an den User und Trafficvermeidung, nicht der Sicherheit. Das Backend ist daher ausführlich zu betrachten.

Durch eine Fehlfunktion besteht eine Gefahr für laufende Projekte bei diversen Kunden. Daraus folgen direkte Schäden und Imageverlust. Fehlerkorrekturkosten werden durch TDD und CI eingegrenzt – Regressionstests und Deployment sind automatisiert.

Testkosten sind ebenso schwierig vorzusehen. Vor allem der geringe Reifegrad des Entwicklungsprozesses erschwert die Schätzung: Bisherige Erfahrungen sind durch den prototypischen Projekt-Charakter nicht vorhanden. Der Testplan kommt zum ersten Mal zum Einsatz. Allerdings wird die Testbarkeit und Modularität der Software durch testgetriebene Entwicklung sichergestellt. Zusätzlich ist die Testinfrastruktur von Anfang an vorhanden und erfüllt alle Anforderungen an Continuous Integration. Die Projektteilnehmer sind mit diesen Werkzeugen vertraut und beherrschen die Grundlagen des Testens nach ISTQB.

Die höchste Priorität haben die Tests des sicherheitskritischen Backends. In der App-Komponente decken die Akzeptanzkriterien Funktionalitäten mit hoher Nutzungshäufigkeit ab. Sie stehen im Mittelpunkt der Kundenwahrnehmung des Produktes und werden entsprechend wichtig eingestuft.

Testaufwand: Eine vollständige Analyse der Aufwände ist nicht möglich und Erfahrungswerte sind noch nicht vorhanden. Diese Herausforderungen stellen ein großes Projektrisiko dar. Zumindest durch bekannte QM-Werkzeuge und hohe Automatisierung soll die Erfolgswahrscheinlichkeit erhöht werden. Den Testaufwand schätzen wir bei diesem Prototypen mit mindestens 50 % des Gesamtentwicklungsaufwands ein.

### **Qualitätsziele und Testabdeckung**

Testendkriterium ist eine Anforderungsüberdeckung von 100 %. Ebenso müssen die sicherheitskritischen Äquivalenzklassentests des Backends zu 100 %.

erfolgreich laufen. Testabbruchkriterien sind fehlgeschlagene Unit- und Akzeptanztests. Fehler der statischen Analyse führen ebenso zum Abbruch, Warnungen oder Schwankungen in den Komplexitäts-Metriken allerdings nicht.

Die nicht-funktionalen Anforderungen des UI5-Design-Guides können nur teilweise durch statische Analysen automatisiert überprüft werden. Gezielte Reviews während der Designphase und bei größeren Änderungen ergänzen die automatische Analyse. Die Performance der App wird im Rahmen der Systemtests überprüft. Die Akzeptanztests werden hierzu mit kurzen Timeouts angepasst.

### Konfigurationsverwaltung

Die Konfigurationsverwaltung erfolgt per Git über die Plattform GitHub. Es gelten die UI5 Guidelines zu Git Commits: Es muss jederzeit nachvollziehbar sein, wer, wann, warum, welche Codezeilen geändert hat.

Eingecheckt wird nicht nur der Programmcode, sondern auch die Dokumentation, Tests und die CI-Toolchain. Das Prinzip *Infrastructure as a Code* ist vollständig umgesetzt. Das bedeutet, die gesamte Build- und Deployment-Umgebung kann auf einem neuen System automatisch wiederhergestellt werden.

Die Abhängigkeiten der Tool- und Framework-Versionen sind in Konfigurationsdateien hinterlegt und werden über die Paketmanager aufgelöst. Über Git werden Zwischenversionen der App gekennzeichnet oder in Branches ausgegliedert. Die Wiederherstellung eines bestimmten Entwicklungsschrittes inklusive Dokumentation, Buildumgebung und Tests ist stets gewährleistet.

Das CI-System stellt die Gültigkeit der Konfigurationsverwaltung sicher. Jede Neuerung in der Versionskontrolle löst einen vollständigen Neuaufbau der Buildumgebung auf dem CI-Server anhand der Konfiguration aus.

### Fehlermanagement

Ein Testprotokoll wird für jeden Build separat auf dem CI-Server vorgehalten. Das gesamte Projektteam hat Zugriff auf das Protokoll. Zusätzlich wird der Build-Status im Repository visualisiert. Das Team ist jederzeit über den Teststatus informiert.

Durch TDD existiert theoretisch kein ungetesteter Code. Die QA-Pipeline wird in einem *Private Build* auf dem Entwicklungsrechner komplett durchlaufen. Noch nicht implementierte Funktionalitäten bzw. die zugehörigen roten Tests werden auch nicht in die globale Versionsverwaltung eingecheckt. Neue Fehlerwirkungen werden vor allem während der höheren Teststufen wie Integrations-, System- und Akzeptanztests auftreten. Denkbar sind auch Inkompatibilitäten mit Browsern und Endgeräten oder Probleme mit dem CI-Prozess selbst.

Da alle Projektartefakte in der Versionsverwaltung eingecheckt werden, ist auch das Fehlermanagement zentral für alle Projektteilnehmer. Das Schema einer Fehlermeldung ist festgelegt und beinhaltet eine eindeutige Fehlernummer, Entdecker, Erfassung und Problembeschreibung. Angaben zur zugehörigen Anforderung, Fehlerquelle und Reproduktion werden wenn möglich eingetragen. Über verschiedene Labels werden für dieses Projekt individuell gesetzt: Fehlerklasse (Bug, Enhancement, Documentation), Priorität (open, wontfix) und Status (new, open, approved, invalid, question, duplicate, works, fixed).

Testfälle und Korrekturen werden über einen Fork eingebracht. Beides kann per Kommentarfunktion diskutiert werden. Ist der Fehler durch einen Testfall nachgewiesen und die Code-Korrektur akzeptiert, schließt ein Kommentar im Commit automatisch das Issue Ticket.

### 3.2.2 Durchführung

Die Testfälle unterteilen sich in verschiedene Teststufen:

- Komponententests der Funktionsbausteine
- Integrationstests zwischen Backend und Gateway
- Isolierter Akzeptanztests der App mit Mockserver
- Systemtest mit App und Gateway

#### Akzeptanztest-Szenario 1: Kundenübersicht

**Given** I start the app

**When** I look at the screen

**Then** I should see the customer list

*and* the customer list should have entries

#### Akzeptanztest-Szenario 2: Projektübersicht

**Given** I start the app

**When** I press on customer 1

**Then** I should be taken to customer 1

*and* I should see the project list of customer 1

*and* the project list should have entries

### Äquivalenzklassentest

Um alle möglichen Eingabewerte für einen Funktionsbaustein zu ermitteln, werden Äquivalenzklassen für die jeweiligen Eingabeparameter gebildet, siehe Tabelle 3.1. Hierbei ist darauf zu achten, dass für jeden Parameter auch

negative Werte getestet werden. Da die Anzahl der Testfälle bei einfacher Multiplikation der möglichen Kombinationen schnell zu einer erheblichen Anzahl ansteigt, werden gültige Äquivalenzklassen in den Testfällen kombiniert. Ungültige Äquivalenzklassen dürfen wiederum nur mit gültigen Äquivalenzklassen kombiniert werden, sodass für jede ungültige Äquivalenzklasse ein eigener Testfall erstellt wird [4, S. 110-115]. Die daraus entstandenen Testfälle sind in Tabelle 3.2 zu sehen. Damit wird eine Äquivalenzklassenüberdeckung von 100 % für den zu testenden Funktionsbaustein erreicht.

Äquivalenzklasse	Parameter	Eingabewert	ÄK gültig?
ÄK1	Kunde	"TKMI"	Ja
ÄK2	Kunde	"tkmi"	Nein
ÄK3	Kunde	-	Nein
ÄK4	Projekt	"TM-SCRUM"	Ja
ÄK5	Projekt	"tm-scrum"	Nein
ÄK6	Projekt	-	Ja
ÄK7	Substring	"TM-SCRUM"	Ja
ÄK8	Substring	"tm-scrum"	Ja
ÄK9	Substring	"asdf"	Nein
ÄK10	Substring	-	Ja
ÄK11	Projekt und Substring	nicht leer	Nein

**Tabelle 3.1:** Äquivalenzklassen für den Funktionsbaustein Z\_SCRUMUI5\_READ\_PROJEKTE

Testfall	getestete ÄK	Kunde	Projekt	Substring	Ergebnis
1	1,6,10	"TKMI"	-	-	Anzeige erfolgt
2	4	"TKMI"	"TM-SCRUM"	-	Anzeige erfolgt
3	7	"TKMI"	-	"TM-SCRUM"	Anzeige erfolgt
4	8	"TKMI"	-	"tm-scrum"	Anzeige erfolgt
5	2	"tkmi"	-	-	Keine Anzeige
6	3	-	-	-	Keine Anzeige
7	5	"TKMI"	"tm-scrum"	-	Keine Anzeige
8	9	"TKMI"	-	"asdf"	Keine Anzeige
9	11	"TKMI"	"TM-SCRUM"	"TM-SCRUM"	Keine Anzeige

**Tabelle 3.2:** Ermittelte Testfälle für den Funktionsbaustein Z\_SCRUMUI5\_READ\_PROJEKTE

**Testfälle OData-Service**

Die relevanten Testfälle (siehe Tabelle 3.3) für den OData-Service werden durch die verwendeten OData-Funktionen bestimmt. Dabei ist zwischen den allgemeinen Testfällen für den kompletten OData-Service, sowie den Funktionen die für jede Entität einzeln getestet werden zu unterscheiden.

Testfall	OData-Funktion
Allgemeine Testfälle:	
TF1	JSON-Format
TF2	Servicedokument
TF3	\$metadata
Testfälle je Entität:	
TF4	\$expand
TF5	Navigation
TF6	Navigation mit \$links
TF7	\$count
TF8	\$orderby
TF9	\$skip
TF10	\$top
TF11	GetEntitySet
TF12	GetEntity
TF13	\$filter

**Tabelle 3.3:** Testfälle - OData-Service

# Kapitel 4

## Implementierung

### 4.1 Backend

#### 4.1.1 Werkzeuge

Für die verschiedenen Abschnitte im Entwicklungsprozess eines Gateway-Services stellt SAP folgende Werkzeuge zur Verfügung.

##### **OData Modeler**

Das grafische Werkzeug zur OData-Service-Modellierung ist als Eclipse-Plugin in den *SAP Mobile Platform Tools* enthalten. Entitäten werden inklusive (Navigations-)Eigenschaften und Abhängigkeiten definiert, siehe Abbildung 4.1.

Die Service-Metadaten werden als XML-Datei exportiert und vom UI5-Mock-Server genauso importiert wie vom Service Builder des SAP Gateway. Eine direkte Verbindung zum SAP-System und Import/Export bestehender Services ist ebenso möglich. Ab der Service-Definition laufen Front- und Backend-Entwicklung unabhängig voneinander.

##### **SAP Gateway Service Builder**

Das zentrale Werkzeug zur Erstellung von OData-Services ist der SAP Gateway Service Builder (Transaktion *SEGW*). Der Service Builder enthält alle Funktionalitäten, die für die Serviceentwicklung benötigt werden [2, S. 187-189].

##### **Gateway-Client**

Zum Testen des OData-Services wird der sogenannte Gateway-Client (Transaktion */IWFND/GW\_CLIENT*) verwendet. Mit ihm lassen sich alle Funktionalitäten des zuvor erstellten OData-Services testen. Testfälle lassen sich



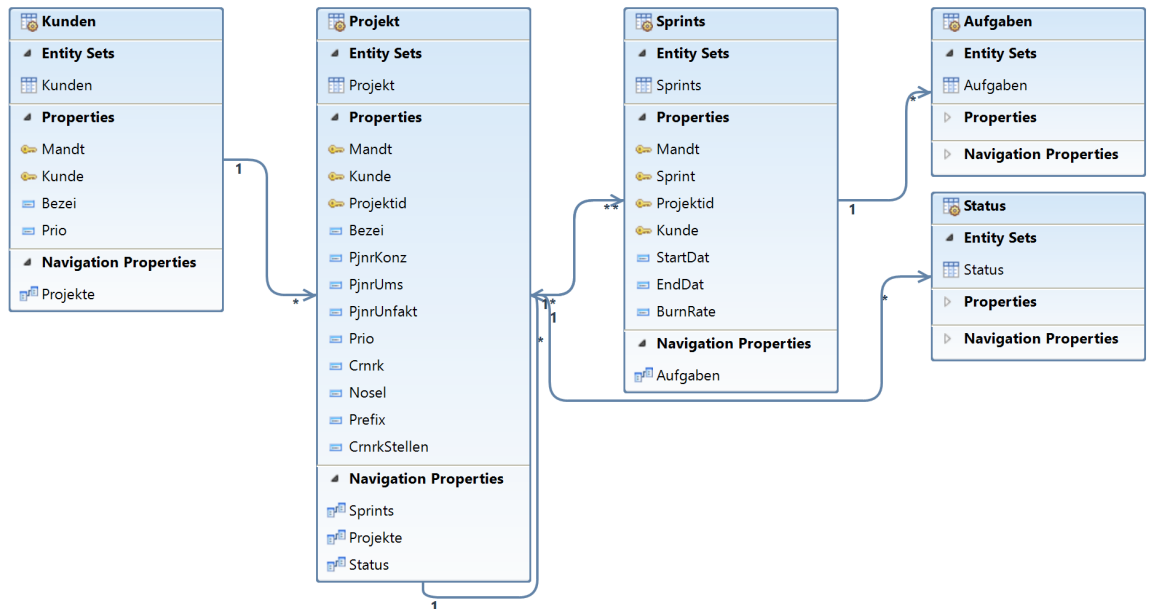


Abbildung 4.1: OData Modeler.

definieren und in Testgruppen zusammenzufassen, um sie später erneut auszuführen [2, S. 190-192].

### Fehlerprotokoll

Zur Fehlerbehandlung gibt es das Fehlerprotokoll unter der Transaktion */IWFND/ERROR\_LOG*. Hier laufen alle Fehler auf, die beim Aufruf von Gateway-Services auftreten. Bei Bedarf ist es dank Integration des Gateway Clients möglich, eine fehlerhafte Abfrage erneut auszuführen [2, S. 192-193].

### Services aktivieren und verwalten

In der Transaktion */IWFND/MAINT\_SERVICE* werden neue Services registriert und aktiviert. Ausgewählte Services lassen sich im Browser oder Gateway-Client testen und das dazugehörige Fehlerprotokoll wird, wenn nötig, geöffnet [2, S. 571-573].

#### 4.1.2 Funktionsbausteine

Für den Zugriff auf Daten aus dem SAP-Backend werden in der zentralen Aufgabenverwaltung derzeit Funktionsbausteine verwendet. Diese werden remotefähig gemacht, um über die RFC-Schnittstelle aufrufbar zu sein. Für die Abfrage der Daten werden die RFC-Funktionsbausteine vom OData-Service

mit entsprechenden Parametern aufgerufen. Die RFC-Funktionsbausteine kommen dann beim Lesen der Kunden-, Projekt-, Sprint- und Aufgabendaten sowie der möglichen Aufgabenstatus der einzelnen Projekte zum Einsatz.

```

FUNCTION z_scrumui5_read_projekte.
*-----
**"Lokale Schnittstelle:
*  IMPORTING
*      VALUE(KUNDEN) TYPE  ZAV_DE_KUNDE
*      VALUE(PROJEKT) TYPE  ZAV_DE_PROJEKT OPTIONAL
*      VALUE(SUBSTRING_BEZEI) TYPE  STRING OPTIONAL
*  TABLES
*      PROJEKTE STRUCTURE  ZAV_PROJEKTE
*-----

DATA: wa_projekte LIKE LINE OF projekte.
"Übergabe Kunde - Alle Projekte eines Kunden (Nur Kunde)
IF kunden <> '' AND projekt = '' AND substring_bezei = ''.
    SELECT * FROM zav_projekte
        JOIN zav_rechte ON zav_projekte~projekt = zav_rechte~projekt
        AND zav_projekte~kunde = zav_rechte~kunde
        INTO CORRESPONDING FIELDS OF TABLE projekte
        WHERE zav_rechte~benutzer = sy-uname
        AND zav_projekte~kunde = kunden
    "Übergabe Kunde und Projekt - Ein bestimmtes Projekt
ELSEIF kunden <> '' AND projekt <> '' AND substring_bezei = ''.
    ...
    "Übergabe Kunde und Substring - Projekte mit Substring in Bezeichnung
ELSEIF kunden <> '' AND projekt = '' AND substring_bezei <> ''.
    SELECT * FROM zav_projekte
        JOIN zav_rechte ON zav_projekte~projekt = zav_rechte~projekt
        AND zav_projekte~kunde = zav_rechte~kunde
        INTO CORRESPONDING FIELDS OF TABLE projekte
        WHERE zav_rechte~benutzer = sy-uname
        AND zav_projekte~kunde = kunden.
    LOOP AT projekte INTO wa_projekte.
        SEARCH wa_projekte-bezei FOR substring_bezei .
        IF sy-subrc <> '0'.
            DELETE TABLE projekte FROM wa_projekte.
        ENDIF.
    ENDLOOP.
ENDIF.
ENDFUNCTION.

```

Listing 3: Z\_SCRUMUI5\_READ\_PROJEKTE

## Entwicklung

In allen Funktionsbausteinen werden Importparameter definiert, die vom OData-Service übergeben werden können. Dieses sind in der Regel die Primärschlüssel der Datenbanktabellen, auf die durch einen Funktionsbaustein zugegriffen wird, um eindeutige Datensätze auslesen zu können. Zusätzlich wird ein Parameter zur Filterung nach Teil-Strings definiert. Die ausgelesenen Daten werden über eine Tabelle, mit der Struktur der ursprünglichen Datenbanktabelle, zurückgegeben.

Im Listing 3 wird der Funktionsbaustein zum Auslesen der Projektdaten dargestellt. Es gibt drei mögliche Szenarien zum Aufruf des Bausteins mit den jeweils nötigen Übergabeparametern:

1. Alle Projekte eines Kunden (KundenID)
2. Ein bestimmtes Projekt des Kunden (KundenID, ProjektID)
3. Projekte des Kunden die Teil-String enthalten (KundenID, ProjektID)

In den ersten beiden Szenarien werden einfache Select-Statements auf die Datenbanktabellen ausgeführt um auf die Daten zuzugreifen. Da mit dem SQL Like Operator im Open SQL die Nichtbeachtung von Groß- und Kleinschreibung nicht möglich ist, wird im letzten Szenario, nach dem Select-Statement, durch alle zurückgegebenen Datensätze gelaufen und überprüft ob der übergebene Teil-String in der Bezeichnung des Projektes vorhanden ist. Hierbei ist es dann egal, ob die Groß- und Kleinschreibung des Datenbankeintrages identisch mit der des Teil-Strings ist.

## Testen

Zum Testen von neuen Funktionsbausteinen gibt es in der ABAP Workbench die Funktion den jeweiligen Funktionsbaustein auszuführen und zu testen. Dazu wählt man in der Drucktastenleiste *Testen/Ausführen (F8)*. Man gelangt auf das Bild *Funktionsbaustein testen: Eingabebild*. Hier werden alle Import-Parameter des Funktionsbausteins angezeigt. Nach Eingabe der Parameter kann der Funktionsbaustein mit *Ausführen (F8)* gestartet werden. Der Funktionsbaustein wird mit den eingetragenen Importparametern ausgeführt und vorhandene Exportparameter oder Tabellen werden nach Ausführung angezeigt.

Es ist möglich Testfälle im Testdatenverzeichnis zu speichern. Das Testdatenverzeichnis erreicht man über den ersten Eingabebildschirm. Alle gespeicherten Testfälle werden mit Erstelldatum und Kurztext wie in Abbildung 4.2 angezeigt und lassen sich hierüber erneut aufrufen. Bei Ausführung eines Regressionstest wird im Ergebnis angezeigt ob sich das Ergebnis seit dem Speichern des Testes verändert hat, siehe Abbildung 4.3. Falls eine Differenz zum gespeicherten Ergebnis vorhanden ist, lässt sich diese anzeigen.

<b>Testdatenverzeichnis: Einzeltests</b>			
Regressiontest    Testergebnis zeigen			
Datensatz-Nummer	Datum	Uhrzeit	Kurztext
1	11.03.2015	14:54:44	TF4 Alle Projekte eines Kunden
2	11.03.2015	14:55:35	TF5 Ein bestimmtes Projekt
3	11.03.2015	14:56:01	TF6 Projekte mit Substring in Bez. upper
4	11.03.2015	14:56:34	TF7 Projekte mit Substring in Bez. lower
5	11.03.2015	14:57:21	NTF4 - Fehlender Parameter Kunde
6	11.03.2015	14:58:33	NTF5 - Kunde nicht vorhanden
7	11.03.2015	14:58:52	NTF6 - Projekt nicht vorhanden
8	11.03.2015	14:59:12	NTF7 Projekt mit substring nicht vorhand

Abbildung 4.2: Funktionsbaustein – Testdatenverzeichnis.

<b>Funktionsbaustein testen: Ergebnisbild</b>	
Testdiff. anz.	
Test für Funktionsgruppe	ZAV_SCRUMUI5
Funktionsbaustein	Z_SCRUMUI5_READ_PROJEKTE
Klein-Groß-Schreibung	<input type="checkbox"/>
Laufzeit:	18.626 Mikrosekunden
Aktueller Test und abgespeicherter Test liefern das gleiche Ergebnis	
RFC-Zielsystem:	
<b>Import-Parameter</b>	<b>Wert</b>
KUNDEN PROJEKT SUBSTRING_BEZEI	TKMI
<b>Tabellen</b>	<b>Wert</b>
PROJEKTE	0 Einträge 9 Einträge
Ergebnis:	

Abbildung 4.3: Funktionsbaustein – Regressionstest.

### 4.1.3 OData-Service

Für die Erstellung eines OData-Services im SAP Gateway gibt es generell unterschiedliche Wege: Zum einen gibt es die Serviceentwicklung mit Advanced Business Application Programming (ABAP), welches flexiblere und effizientere Services ermöglicht. Diese erfordert allerdings spezielles ABAP-Know-how.

Alternativen dazu sind die Servicegenerierung durch den RFC/Business

Object Repository (BOR)-Generator, die Redefinition oder die Model Composition von bereits vorhandenen Gateway-Services. Die Servicegenerierung führt zu schnelleren Ergebnissen, ist durch eingeschränkte Rechte allerdings oft nicht weiter anpassbar.

Im Normalfall rechtfertigen die Vorteile durch die klassische ABAP-Service-Entwicklung den höheren Aufwand. Die Möglichkeit der Servicegenerierung wird interessant, sobald es geeignete Datenquellen für die automatische Generierung gibt [2, S. 181].

Generell wird ein inkrementeller Serviceerstellungsprozess empfohlen, in dem der Service bzw. Teile davon direkt ausgeführt und getestet werden. Danach kann der Service so lange verändert werden, bis schließlich alle Anforderungen erfüllt werden [2, S. 184].

### Entwicklung

Die Entwicklung eines Gateway-Services lässt sich in drei Phasen einteilen:

1. Definition des Datenmodells
2. Serviceimplementierung
3. Serviceverwaltung

Zur *Definition des Datenmodells* im SAP Gateway Service Builder gibt es unterschiedliche Varianten. Dazu zählen unter anderem die manuelle Deklaration des Datenmodells im Service Builder. Hierbei werden die Entitätstypen, Assoziationen und Assoziationsmengen manuell angelegt.

Alternativ kann ein bereits vorhandenes Datenmodell importiert werden, welches z. B. mit dem OData Modeler erstellt wurde. Falls eine bereits vorhandene Datenstruktur aus einem SAP-System genutzt werden soll, können die Entitätstypen über einen DDIC-Import oder über den Import von RFC/BOR-Schnittstellen generiert werden [2, S. 194-195].

In unserem Fall kommt der Import über RFC-Funktionsbausteine wie in Abbildung 4.4 zum Einsatz, da diese auch für den Zugriff auf Daten aus dem SAP-Backend genutzt werden. Während des Imports des RFC-Funktionsbausteins müssen die Primärschlüssel der Entitätstypen festgelegt werden.

Unter den Eigenschaften der Entitätstypen lassen sich die einzelnen Attribute der Entitätstypen anzeigen und deren Eigenschaften ändern. So wird etwa definiert, ob ein bestimmtes Attribut sortier- oder filterbar ist. Diese Eigenschaften werden im Service-Metadatendokument hinterlegt. Diese haben allerdings nur einen informativen Charakter: Es erfolgt keine weitere Überprüfung durch das Gateway-Framework. So können Attribute sortiert werden, für die keine Sortier-Eigenschaft gesetzt wurde.

Als nächstes müssen die Entitätsmengen definiert werden, die im Regelfall in einer 1:1-Beziehung zu den Entitätstypen stehen. Der spätere Zugriff über

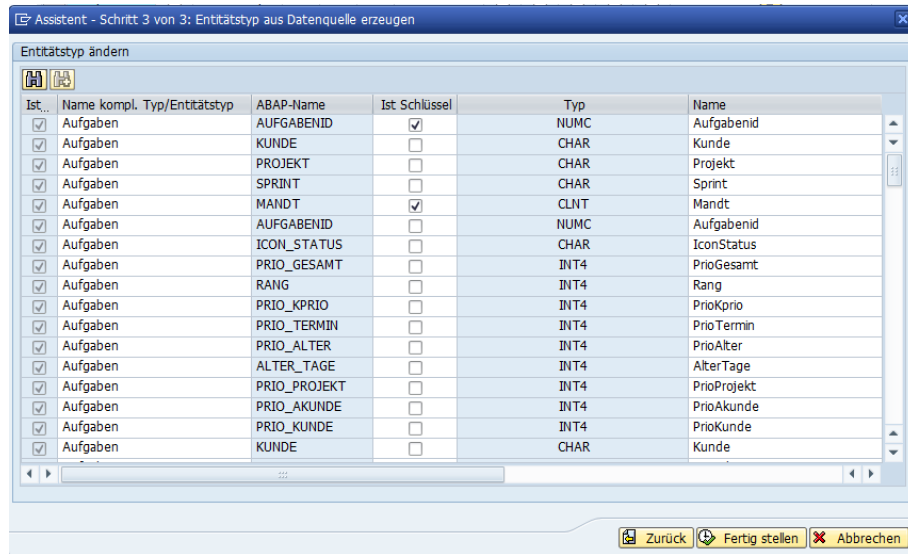


Abbildung 4.4: Import-RFC-Funktionsbaustein [2, S. 45].

den OData-Service läuft immer über die Entitätsmengen und nicht über die Entitätstypen. Auch hier gibt es einige Attribute zu pflegen, z. B., ob ein Eintrag eines Entitätstypen angelegt, aktualisiert oder gelöscht werden darf, oder ein Filter für die Abfrage der Entitätsmenge zwingend erforderlich ist. Wie auch die Annotationen der Entitätstypen sind auch diese nicht zwingend bindend und erforderlich, sollten jedoch passend zum tatsächlichen Funktionsumfang des OData-Services gepflegt werden. So ist es für Service-Konsumenten einfacher herauszufinden, welchen Funktionsumfang der OData-Service unterstützt [2, S. 237-240].

Anschließend wird der Service im SAP-System registriert. Hier werden die Laufzeitobjekte, die für den Gateway-Service benötigt werden, generiert [2, S. 198]. Danach geht es an die *Serviceimplementierung*. Hier gibt es die Auswahl zwischen der Implementierung durch ABAP-Programmierung oder das Mappen von RFC/BOR-Schnittstellen [2, S. 201-202].

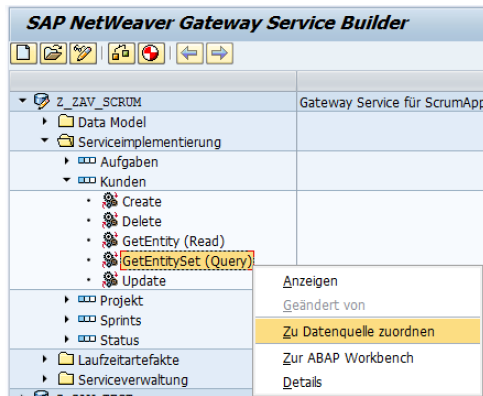


Abbildung 4.5: RFC-Mapping.

diesem Zeitpunkt ist lediglich das Abrufen von Daten ohne zusätzliche Funktionen möglich. Um die Filterfunktionen des OData-Services nutzen zu können, müssen die Import-Parameter der Funktionsbausteine zugeordnet werden.

Dafür wird eine neue Zeile mit der Entitätsmenge des Importparameters hinzugefügt – In diesem Fall *Kunde*. Dieser wird dem Import-Parameter aus der Datenquelle zugeordnet. Jetzt kann nach bestimmten Kunden gefiltert werden. So lassen sich Services mit Grundfunktionen relativ einfach durch die Servicegenerierung erstellen. Sobald Funktionen wie das Filtern nach Teil-Strings benötigt werden, ist eine Zusatzimplementierung durch ABAP-Programmierung notwendig.

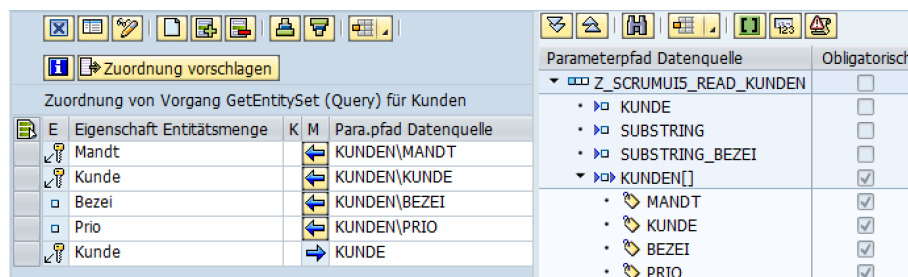


Abbildung 4.6: RFC-Mapping – Zuordnung der Attribute.

Falls der Service durch ABAP-Programmierung erstellt werden soll, müssen die Methoden für den Vorgang in der ABAP-Workbench redefiniert werden. Die bei der Servicegenerierung automatisch erstellte Methode wird hier selbst implementiert. Alle Parameter, die vom OData-Service an das Gateway gesendet werden, müssen berücksichtigt und die entsprechenden Funktionen implementiert werden.

Um die Attribute einer Entitätsmenge mit einem RFC-Funktionsbaustein zu mappen, wählt man *Zu Datenquelle zuordnen* für den gewünschten Vorgang einer Entitätsmenge, hier: GetEntitySet der Entitätsmenge Kunden, siehe Abbildung 4.5.

Nach Angabe des Funktionsbausteins können die Attribute der Quelle zugeordnet werden.

Dies funktioniert manuell oder über den Button *Zuordnung vorschlagen* wie in Abbildung 4.6. Zu

In Tabelle 4.1 gibt es eine kurze Übersicht, der vom SAP Gateway unterstützen OData-Query-Optionen, wovon einige eine zusätzliche Implementierung benötigen.

OData-Query-Optionen	Zusätzliche Implementierung notwendig
\$select	Nein
\$count	Nein
\$expand	Nein
\$format	Nein
Read \$links	Nein
\$value	Nein
\$orderby	Ja
\$top	Ja
\$skip	Ja
\$filter	Ja
\$inlinecount	Ja
\$skiptoken	Ja

**Tabelle 4.1:** Unterstützte OData-Query-Optionen im SAP Gateway aus [22]



```

METHOD kunden_get_entityset.
DATA:  ls_headerdata TYPE zav_kunden,
       lt_headerdata TYPE STANDARD TABLE OF zav_kunden,
       ls_project LIKE LINE OF et_entityset.

DATA:  ls_filter_select_options TYPE /iwbep/s_mgw_select_option,
       ls_select_option TYPE /iwbep/s_cod_select_option,
       ivfiltersubstring TYPE string.

LOOP AT it_filter_select_options INTO ls_filter_select_options.
  IF ls_filter_select_options-property EQ 'Bezei'
  OR ls_filter_select_options-property EQ 'Kunde'.
    LOOP AT ls_filter_select_options-select_options INTO ls_select_option.
      IF ls_select_option-option = 'CP'.
        ivfiltersubstring = ls_select_option-low.
      ENDIF.
    ENDLOOP.
  ENDIF.
ENDLOOP.

REPLACE ALL OCCURRENCES OF '*' IN ivfiltersubstring WITH ''.

CALL FUNCTION 'Z_SCRUMUI5_READ_KUNDEN'
  EXPORTING
    substring_bezei = ivfiltersubstring
  TABLES
    kunden          = lt_headerdata.

LOOP AT lt_headerdata INTO ls_project.
  APPEND ls_project TO et_entityset.
ENDLOOP.

ENDMETHOD.

```

**Listing 4:** KUNDEN\_GET\_ENTITYSET

In Listing 4 wird, falls in der Abfrage angegeben, ein Teil-String der Kundenbezeichnung in einer Variablen gespeichert. Diese wird anschließend an den Funktionsbaustein übergeben. Je nach Abfrage werden alle Kunden oder nur Kunden mit Teil-String im Namen zurückgegeben. Zum Abschluss werden die zurück erhaltenen Daten in die Tabelle *ET\_ENTITYSET* geschrieben und an den OData-Service übergeben.

Nach Implementierung des Services muss dieser noch über die *Serviceverwaltung* beim Gateway-Server registriert und aktiviert werden. Anschlie-

ßend kann der OData-Service getestet werden. Sind voneinander abhängige Entitätstypen im Service vorhanden, werden die Relationen mit den entsprechenden Kardinalitäten zwischen den Entitätstypen in den Assoziationen wie in Abbildung 4.7 definiert. Die zuvor erstellten Assoziationen werden anschließend den Entitätstypen als Navigationseigenschaften zugewiesen, siehe Abbildung 4.8.

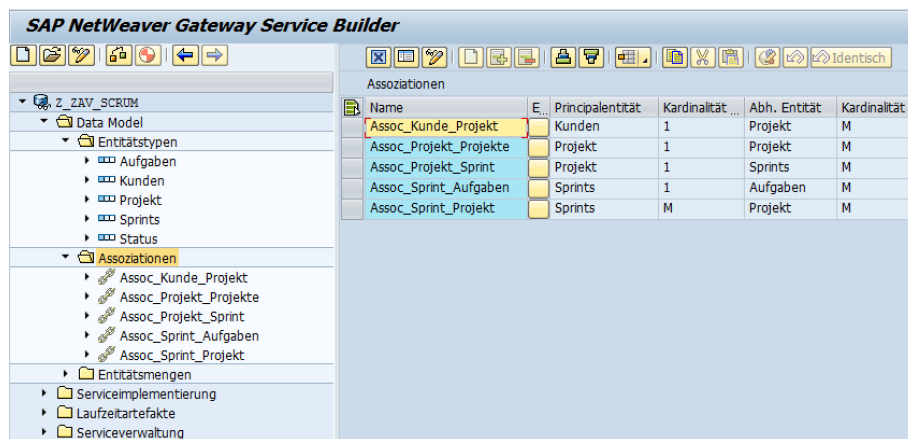


Abbildung 4.7: Assoziationen zwischen den Entitätstypen.

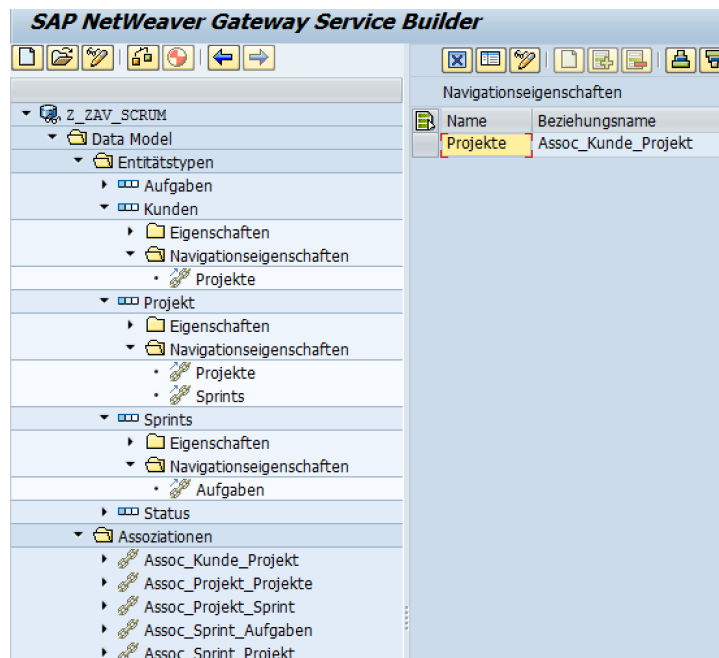


Abbildung 4.8: Navigationseigenschaften zwischen den Entitätstypen.

**SAP NetWeaver Gateway Client - Aus Datenbank auswählen**

Neu auswählen

Request-Daten Erwarteten Status setzen Ausführen

**Testfälle**

Zeile	Testgru...	Testfall	Metho...	Erwartet...	Request-URI
1	SCRUMUIS	\$format=json	GET	200	/sap/opu/odata/SAP/Z_ZAV_SCRUM_SRV/?\$format=json
2		\$metadata	GET	200	/sap/opu/odata/SAP/Z_ZAV_SCRUM_SRV/\$metadata
7		Kunden - > Projekte	GET	200	/sap/opu/odata/SAP/Z_ZAV_SCRUM_SRV/Kunden(Mandt='001',Kunde='TKMI')/Projekte
8		Kunden - > Projekte (links)	GET	200	/sap/opu/odata/SAP/Z_ZAV_SCRUM_SRV/Kunden(Mandt='001',Kunde='TKMI')/links/Projekte
3		Kunden - \$count	GET	200	/sap/opu/odata/SAP/Z_ZAV_SCRUM_SRV/Kunden/\$count
4		Kunden - \$orderby	GET	200	/sap/opu/odata/SAP/Z_ZAV_SCRUM_SRV/Kunden?\$orderby=Kunde
5		Kunden - \$skip	GET	200	/sap/opu/odata/SAP/Z_ZAV_SCRUM_SRV/Kunden?\$skip=2
6		Kunden - \$top & \$orderby desc	GET	200	/sap/opu/odata/SAP/Z_ZAV_SCRUM_SRV/Kunden?\$top=2&\$orderby=Kunde desc
9		Kunden - Alle Kunden	GET	200	/sap/opu/odata/SAP/Z_ZAV_SCRUM_SRV/Kunden
10		Kunden - Ein Kunde	GET	200	/sap/opu/odata/SAP/Z_ZAV_SCRUM_SRV/Kunden(Mandt='001',Kunde='ABAT')
11		Kunden - filter substringof	GET	200	/sap/opu/odata/SAP/Z_ZAV_SCRUM_SRV/Kunden?\$filter=substringof('S',Bezel)

Abbildung 4.9: OData-Testfalldatenbank.

**SAP NetWeaver Gateway Client - Mehrfachtest**

Test Status: NOT OK

Total 3

Successful 2

Erroneous 1

Duration 00:00:12

Request-Daten Response-Daten Fehlerprotokoll Erwarteten Status setzen Re-Test

**Testergebnisse**

Zeile	Status	Datum	Uhrzeit	Testfall	Ant...	Erwartet...	Fehlertext
1	✓	12.03.2015	16:33:29	Projekte - Alle Projekte eines Kunden (FB TF4)	200	200	
2	✓		16:33:37	Projekte - Ein bestimmtes Projekt (FB TF5)	200	200	
3	✗		16:33:40	Projekte - NTF4 - Projekte ohne Kunde	404		HTTP Status Code 404 ( Not Found )

Abbildung 4.10: OData-Mehrfachtest.

## Testen

Ist der OData-Service erstellt und registriert, geht es an das Testen mit dem SAP Gateway Client. Wie bereits in Abschnitt 4.1.1 erwähnt, lassen sich die Testfälle für den OData-Service in Testgruppen speichern, und jederzeit wieder abspielen, siehe Abbildung 4.9. Das Abspielen geschieht in einer sequenziellen Reihenfolge, um so komplexe Szenarien mit Erstellung, Update, Lesen und Löschen von Daten zu testen. Zu jedem abgesicherten Testfall lässt sich ein erwarteter HTTP-Statuscode definieren, wie in etwa der Code 200 für den Status *OK* oder 4xx für Client-Fehler [2, S. 555-556].

Nach Ausführung der Testfälle gibt es eine Übersicht wie in Abbildung 4.10, inklusive erwartetem und tatsächlichem HTTP-Statuscode, sowie möglichem Fehlertext. Durch die Integration des Fehlerprotokoll lässt sich über die Auswahl eines fehlgeschlagenen Tests direkt in das Fehlerprotokoll wechseln, um die Fehlerursache zu ermitteln.

Zusätzlich kann im Browser die Query-Option *sap-ds-debug=true* genutzt werden, um Antworten des Services im Browser als HTML-Seite anzeigen zu

lassen. Neben der Rückantwort gibt es zusätzliche Informationen wie z. B. die Laufzeitanalyse der ABAP Methoden. Falls Fehler im Service auftreten, erscheint der Tab *Stacktrace*, in dem die auftretende Exception angezeigt wird [2, S. 555-556].

## 4.2 CI-Toolchain

### 4.2.1 Paketmanager

Die Paketmanager sind existenzieller Bestandteil der CI-Toolchain – Das Prinzip *Infrastructure as Code* ist nur mit ihrer Hilfe umzusetzen: Die Abhängigkeiten müssen nicht mehr in die Versionskontrolle eingecheckt werden, stattdessen werden die benötigten Versionen bei Bedarf extern nachgeladen.

#### Node.js

Node.js ist eine JavaScript-Laufzeitumgebung und für diverse Plattformen verfügbar. Basis ist die schnelle JavaScript-Engine Google V8, die um Techniken wie Nebenläufigkeit und nicht-blockierende I/O-Zugriffe erweitert wurde. Node.js kann eine große Zahl an gleichzeitigen Netzwerkverbindungen verwalten, ein zusätzlicher HTTP-Server ist nicht erforderlich. Ursprünglich sollte Node.js den einfachen Betrieb von Echtzeit-Webanwendungen ermöglichen.

Für dieses Projekt nutzen wir Node.js nicht als Server, sondern machen uns das stark gewachsene Ökosystem in Form des Node.js Package Managers *npm* zu nutze. *npm* ist dabei zum einen ein lokales Tool zur Paketverwaltung, das direkt mit Node.js ausgeliefert wird, zum anderen gibt es ein zentrales Repository mit downloadbaren Paketen.

Diese Prinzip fördert eine hohe Modularisierung: JavaScript-typisch existieren viele kleine Tools, die eine bestimmte Aufgabe lösen, siehe Unterabschnitt 2.1.3. Alle im Projekt genutzten Entwicklungs- und Testtools beziehen wir über *npm*. Die Installation erfolgt nicht global, sondern lokal im Projektverzeichnis *node\_modules*. Für verschiedene Projekte kann das gleiche Tool ohne Probleme in verschiedenen Versionen installiert werden.

Formal ist auch unser UI5-Projekt selbst ein Node.js-Paket. Die Metadaten in der Datei *package.json*, siehe Listing 5, beschreiben es mit Namen und Version. Wir definieren zusätzlich alle Entwicklungsabhängigkeiten. Über den Befehl *npm install* werden diese automatisch in der angegebenen Version installiert. Auch weitere, indirekte Abhängigkeiten werden aufgelöst. Ein neues Tool wird über *npm install package --save-dev* installiert und automatisch, als weitere Abhängigkeit für zukünftige Installationen, gespeichert.

Der Zusatz *private* in der Konfiguration stellt sicher, dass wir das Paket nicht unabsichtlich über *npm* veröffentlichen. Zusätzlich können private Repositories erstellt werden. So ist z. B. eine firmeninterne, projektübergreifende Nutzung von Komponenten möglich.

Als Best Practice haben sich feste Versionen der Entwicklungsabhängigkeiten erwiesen. Im Gegensatz zu einem echten Node.js-Projekt sind sie bei uns nur ein Werkzeug und müssen keine breite Kompatibilität mit externen Abhängigkeiten eines veröffentlichten Pakets gewährleisten. Eine einmal funktionierende Konfiguration sollte beibehalten werden, um den Testaufwand der Infrastruktur gering zu halten.

```
{
  "name": "scrumui5",
  "version": "0.0.1",
  "private": true,
  "devDependencies": {
    "bower": "^1.3.12",
    "eslint": "^0.15.0",
    "eslint-plugin-openui5": "^0.1.0",
    "grunt": "^0.4.5",
    "karma": "~0.12.8",
    "karma-openui5": "~0.2",
    "karma-qunit": "~0.1"
  }
}
```

**Listing 5:** package.json (gekürzt)

## Bower

Der zweite Paketmanager Bower ist nur für Frontend-Frameworks zuständig. In diesem Fall installiert er die UI5-Kernkomponenten *sap.ui.core*, *sap.m* und das Standardtheme *sap\_bluecrystal* im Projektordner unter *bower\_components*. Sie sind zwar als Entwicklungsabhängigkeiten aufgeführt, wir nutzen sie aber gleichzeitig für das Release der App. Ein Grunt-Task kopiert die minified-Versionen in den Target-Ordner. Der Verwaltungs- und Testaufwand wird minimiert, weil für Entwicklung und Release die gleiche Version genutzt wird. Im Gegensatz zum offiziellen OpenUI5-Paket sind beim Bezug über Bower außerdem die Test-Frameworks QUnit und OPA5 enthalten.

Die automatische Paketinstallation wird über *bower install* angestoßen. Auch die Paket-Verwaltung erfolgt synchron zu npm zentral über die Datei *bower.json*, siehe Listing 6. Im Unterschied zu npm unterstützt Bower keine geschachtelten Abhängigkeiten. Jedes Framework existiert nur einmal im Projekt – Der Verzeichnisbaum hat nur eine Ebene. Bower arbeitet dadurch schneller als npm. Die einfache Struktur zwingt außerdem dazu, eine kompatible Framework-Kombination zu finden. Für den Endbenutzer führt dies zu schnelleren Ladezeiten, da nur jeweils eine Version heruntergeladen werden muss.

Mittelfristig wird Bower durch den mächtigeren npm abgelöst. Der Pflegeaufwand für zwei Paketmanager ist hoch. Viele Projekte haben dies erkannt, und bieten auch ihre Frontend-Frameworks per npm an. Bei OpenUI5 ist dies aber noch nicht der Fall.

```
{
  "name": "scrumui5",
  "version": "0.0.1",
  "private": true,
  "devDependencies": {
    "openui5-sap.ui.core": "openui5/packaged-sap.ui.core",
    "openui5-sap.m": "openui5/packaged-sap.m",
    "openui5-sap_bluecrystal": "openui5/packaged-sap_bluecrystal"
  }
}
```

**Listing 6:** bower.json (gekürzt)

#### 4.2.2 Statische Analyse mit ESLint

Da JavaScript nicht kompiliert wird, werden Syntax-Fehler erst während der Ausführung erkannt. Die statische Code-Analyse setzt schon vorher, während der Bearbeitung im Editor, an. Häufige Fehler oder sogenannte Anti-Patterns (nicht empfohlene Vorgehensweisen) sind global deklarierte oder nie genutzte Variablen. Außerdem lassen sich bestimmte Code-Styleguides durchsetzen.

ESLint ist ein Open-Source-Projekt auf Node.js-Basis. Die Installation erfolgt entsprechend per npm. ESLint bringt bereits ein vordefiniertes Regelset mit. Im Gegensatz zu JSLint oder JSHint ist dieses jedoch vollständig frei konfigurierbar. In der Datei *.eslintrc* werden die vorhandenen Regeln als Warnung oder Fehler eingestuft – oder ignoriert. Globale Variablen sind frei definiert oder umgebungsspezifisch als gültig deklariert, z. B. im Node.js- oder jQuery-Kontext.

Wir übernehmen das Regelset aus OpenUI5. Zusätzlich nutzen wir das Plugin *eslint-openui5*, das eine weitere UI5-spezifische Regel für Zeilenenden hinzufügt. Besonderheit: Weder das UI5-Framework, noch die Best-Practice-Beispiele sind mit den Regeln konform. Zum Beispiel das mittlerweile obligatorische *'use strict'*; fehlt. Das von SAP veröffentlichte Regelset ist eher Ziel, als aktueller Standard. Entsprechend unterziehen wir das Framework keiner statischen Prüfung. Unsere Eigenentwicklungen sind allerdings ESLint-konform, um zukünftige Kompatibilität zu gewährleisten.

### 4.2.3 Testtreiber Karma

Karma ist ein JavaScript-Testtreiber auf Node.js-Basis und aus dem Google-Projekt AngularJS hervorgegangen. Er ist das Verbindungsglied zwischen Test-Frameworks (QUnit, OPA5), CI-Tool (Grunt/Jenkins) und Browser (Desktop, Mobil, Headless, ...). Die Architektur ist modular: Verbindungsfunktionen übernehmen Plugins, sogenannte Karma-Adapter.

Installiert wird karma per npm. Anschließend kann die Erstkonfiguration über *karma init karma.conf.js* gestartet werden. Ein Assistent führt zu einer Minimalkonfiguration wie in Listing 7. Wir benötigen die Test-Frameworks QUnit und OPA5. Außerdem speichern wir den Pfad zum OpenUI5-Framework und aktivieren dessen Mockserver.

Beim Start per *karma start*, bzw. über die IDE oder Jenkins/Grunt, stellt Karma bestimmte Dateien über seinen integrierten Server bereit. Die verschiedenen File-Anweisungen in der Konfiguration schließen OpenUI5, unsere App und die Tests ein (*served*). Der Parameter *watched* überwacht das angegebene Verzeichnis. Ändern sich App-Code oder Testdaten, wird ein erneuter Testlauf angestoßen. *included* markiert die Verzeichnisse mit Testskripts.

Die weiteren Parameter aktivieren eine Code-Coverage-Protokollierung, automatische Regressionstests und den zu öffnenden Browser. Möglich wären hier statt Chrome auch diverse andere Desktop-Browser, mobile Geräte und ein Test ohne grafische Oberfläche per PhantomJS. *singleRun* ist eine spezielle CI-Option: Karma beendet Server und Browser automatisch nach einem Testlauf.

```
module.exports = function (config) {
  config.set({
    frameworks: ['qunit', 'openui5'],
    openui5: {
      path: 'bower_components/openui5-sap.ui.core/resources/sap-ui-core.js',
      useMockServer: true
    },
    files: [{
      pattern: 'bower_components/openui5/**/*.js',
      served: true,
      included: false,
      watched: false
    }, {
      pattern: 'app/**/*.js',
      served: true,
      included: false,
      watched: true
    }, {
      pattern: 'test/**/*.js',
      served: true,
      included: true,
      watched: true
    }],
    preprocessors: {
      'app/**/*.js': ['coverage']
    },
    reporters: ['progress', 'coverage'],
    autoWatch: false,
    browsers: ['Chrome'],
    singleRun: true
  });
};
```

**Listing 7:** karma.conf.js – Minimalkonfiguration

Speziell für unser UI5-Projekt benötigen wir den Adapter *karma-openui5*<sup>1</sup>. Er lädt OpenUI5 aus dem bereits angegebenen Verzeichnis. Zusätzlich werden wie in Listing 8 die weiteren Bibliotheken und Themes bereitgestellt. Die Konfiguration erfolgt analog zum UI5-Bootstrapping.

---

<sup>1</sup><https://github.com/SAP/karma-openui5>



```

openui5 : {
  config : {
    theme : 'sap_bluecrystal',
    libs : 'sap.m',
    resourceroots : {
      'sap.m' : '/base/bower_components/openui5-sap.m/resources/sap/m',
      'de.abat.scrumui5.test' : '/base/test',
      'de.abat.scrumui5' : '/base/app'
    },
    themeroots : {
      'sap_bluecrystal' : '/base/bower_components/bluecrystal/resources'
    }
  },
}

```

**Listing 8:** karma.conf.js – Ausschnitt OpenUI5

Der Mockserver wird wie in Listing 9 konfiguriert. *rootURI* entspricht der OData-Service-Adresse, die unsere App aufruft. Der Mockserver fängt entsprechende Aufrufe ab und stellt einen, anhand der *metadata.xml* definierten, Service bereit. Diese kann per OData Modeler erstellt, oder von einem bereits verfügbaren Service exportiert werden.

Unter *mockdataSettings* ist das Verzeichnis der Mock-Daten angegeben, also die Objekte, die der Server zurückliefert. In diesem Projekt sind das z. B. *Kunden.json* oder *Projekt.json*, die wir aus dem laufenden Service exportiert haben. Alternativ erzeugt der Mockserver auch Zufallsdaten aus der Spezifikation.

```

mockserver : {
  config : {
    autoRespond : true
  },
  rootUri : 'http://host:port/sap/opu/odata/sap/Z_ZAV_SCRUM_SRV/',
  metadataURL : '/base/test/service/metadata.xml',
  mockdataSettings : '/base/test/service'
}

```

**Listing 9:** karma.conf.js – Ausschnitt Mockserver

Zur Zeit (Stand: 12.03.2015) ist der Adapter karma-openui5 nicht ohne Modifikation in unserer Toolchain lauffähig. Um die Mockserver-Funktion zu nutzen, muss die Datei *./node\_modules/karma-openui5/lib/mockserver.js* in Zeile 7 erweitert werden. Listing 10 zeigt den aktuellen Workaround. Das Problem ist gemeldet und in Klärung.

```
if (typeof module === undefined){  
    var module = undefined;  
}
```

Listing 10: mockserver.js Erweiterung

#### 4.2.4 Hybrid-App per PhoneGap Build

PhoneGap Build<sup>2</sup> ist ein Cloud-Service von Adobe um mobile Web-Anwendungen auf Basis des Open Source Toolkits Apache Cordova in der Cloud zu kompilieren.

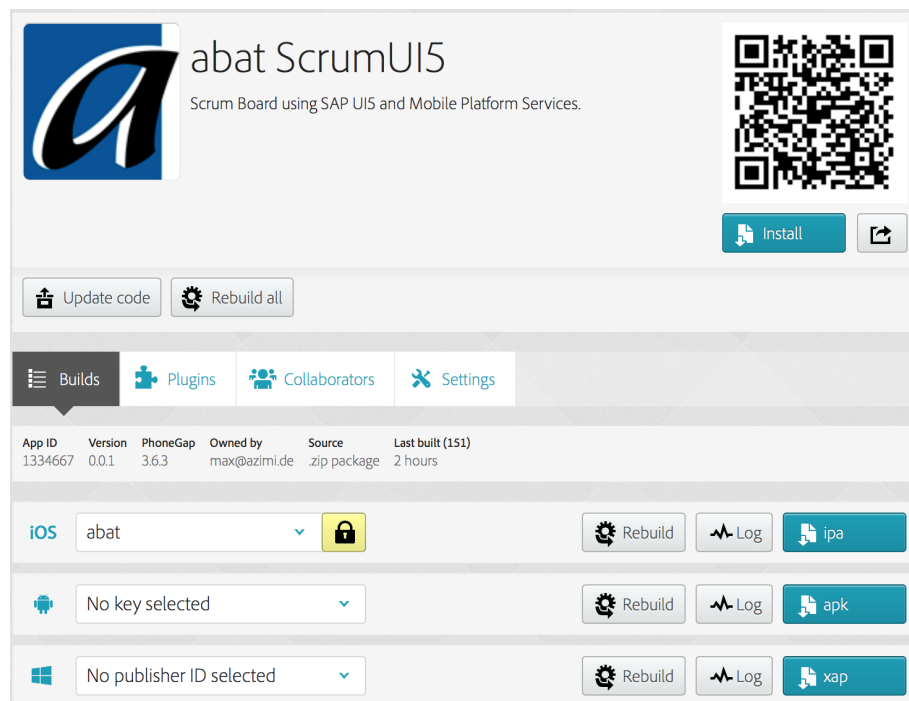


Abbildung 4.11: PhoneGap-Build-Webinterface.

Die Konfiguration des PhoneGap Builds läuft über die Weboberfläche, die auf Abbildung 4.11 zu sehen ist. Der Quellcode der Web-Applikation kann als ZIP-Archiv gepackt und hochgeladen oder automatisch aus einem Git-Repository geladen werden. Anschließend wird die Anwendung für verschiedene Betriebssysteme direkt kompiliert und signiert. Für die Signierung der iOS-App müssen die entsprechenden Entwicklerzertifikate hinterlegt werden.

---

<sup>2</sup><http://build.phonegap.com>

Nach erfolgreichem Build der Anwendung können die Installationsdateien für die verschiedenen Plattformen heruntergeladen und für die weitere Verteilung genutzt werden. Zudem ist es möglich, die Anwendung über einen Link zu verteilen. Dieser installiert auf mobilen Geräten automatisch die App.

Auf native Gerätefunktionen wird durch verschiedene Plugins zugegriffen, sodass Kamera oder GPS-Koordinaten auch für Web-Apps zugänglich sind. Durch das Kompilieren in der Cloud wird kein spezielles Software Development Kit (SDK) für die verschiedenen Plattformen wie iOS und Android benötigt. Es reicht eine Entwicklungsumgebung für Webanwendungen.

#### 4.2.5 JS Task Runner Grunt

Um die Vielzahl an Werkzeugen für unseren automatischen Build aufzurufen, kommt der sogenannte JavaScript Task Runner Grunt<sup>3</sup> zum Einsatz. Einzige Aufgabe ist die Abarbeitung der einzelnen Tasks, die in der Grunt-Konfiguration (siehe Listing 11) definiert werden [16, S. 59-61]. Die eigentlichen Funktionalitäten sind in diverse Grunt-Plugins ausgelagert, die wir im Folgenden erläutern.

##### **grunt-contrib-uglify**

Dies ist ein Grunt-Plugin zum Ausführen des UglifyJS-Toolkits. Hiermit wird der JavaScript-Code komprimiert, Variablennamen werden, wenn möglich, verkürzt und vorhandene Kommentare gelöscht. Die minified-JavaScript-Files besitzen typischerweise nur noch 30–40 % der Ursprungsgröße.

##### **grunt-eslint**

Überprüft den JavaScript-Code auf Einhaltung der vereinbarten Regeln zur statischen Code-Analyse per ESLint, einer Weiterentwicklung von JSHint. Hierfür stellt SAP im OpenUI5-Repository eine ESLint-Konfigurationsdatei zur Verfügung, die in eigenen Projekten verwendet werden kann [20].

##### **grunt-karma**

Dieses Plugin ermöglicht es, Karma während des Build-Prozesses aufzurufen. Über separate Karma-Tasks werden die Unit- und Akzeptanztests ausgeführt. [16, S. 129-130].

##### **grunt-contrib-copy**

Verschiebt Dateien, z. B. vom Source- in den Destination-Ordner.

---

<sup>3</sup><http://gruntjs.com/>

```

module.exports = function (grunt) {
  grunt.initConfig({
    uglify: {
      min: {
        options: {
          mangle: true,
          banner: '/*! \n'
            + ' * <%= pkg.description %> v<%= pkg.version %>\n'
            + ' * Copyright (c) 2015 abat AG\n'
            + ' * Date: <%= grunt.template.today("yyyy-mm-dd") %>\n'
            + ' */\n'
        },
        files: grunt.file.expandMapping(['target/**/*.js',
          '!target/resources/**/*.js'], '', {
            rename: function (destBase, destPath) {
              return destBase + destPath.replace('.js', '.min.js');
            }
          })
      }
    },
    eslint: {
      options: {
        configFile: '.eslintrc',
        ignoreFile: '.eslintignore',
        quiet: false,
        format: 'checkstyle',
        outputFile: 'test-reports/eslint-cov.xml'
      }, target: ['app/**/*.js', 'test/**/*.js']
    }
  });

  grunt.loadNpmTasks('grunt-contrib-uglify');
  grunt.loadNpmTasks('grunt-eslint');

  grunt.registerTask('complete', ['clean', 'eslint', 'plato',
    'karma:unit:start', 'copy', 'uglify:min', 'compress', 'phonegap-build']);
};

```

Listing 11: Auszug Gruntfile.js

**grunt-contrib-compress**

Komprimiert den Target-Ordner in einer ZIP-Datei, damit diese anschließend vom PhoneGap-Build-Plugin weiterverarbeitet wird.

### grunt-phonegap-build

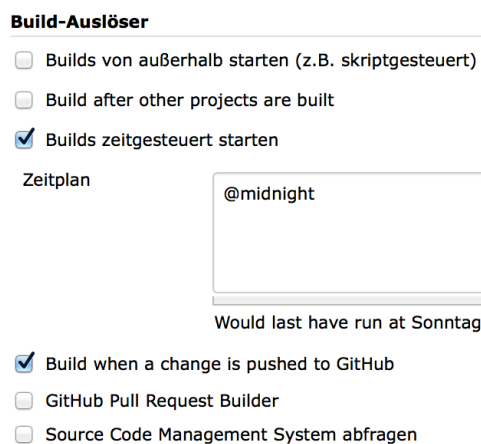
Lädt das in einer ZIP-Datei archivierte Projekt über ein Application Programming Interface (API) zu PhoneGap Build. Anschließend löst es einen neuen Build aus und signiert die Installationsdateien.

### grunt-plato

Neben der statischen Code-Analyse durch ESLint gibt es weitere Analysen des Quelltextes durch das plato-Plugin. Es stellt verschiedene Metriken zur Komplexität und Wartungsfreundlichkeit des Codes bereit.

## 4.2.6 Jenkins

Als Continuous-Integration-Server wird Jenkins<sup>4</sup> eingesetzt. Für die Implementierung verschiedener Werkzeuge im Continuous-Integration-Prozess werden zusätzliche Plugins wie folgt verwendet:



**Build-Auslöser**

- ☐ Builds von außerhalb starten (z.B. skriptgesteuert)
- ☐ Build after other projects are built
- ☒ Builds zeitgesteuert starten

Zeitplan: @midnight

Would last have run at Sonntag

- ☒ Build when a change is pushed to GitHub
- ☐ GitHub Pull Request Builder
- ☐ Source Code Management System abfragen

Abbildung 4.12: Build-Auslöser.

### GitHub-Plugin

Zur Versionsverwaltung der Projektdaten wird GitHub verwendet. Das Plugin stößt den Build-Prozess bei Code-Änderungen im Repository automatisch an. Ebenfalls möglich ist eine zeitgesteuerte Ausführung oder ein manueller Start wie in Abbildung 4.12. So ist gewährleistet, dass der Build inkl. Tests mit der aktuellen Version des Codes durchgeführt wird.

<sup>4</sup><https://jenkins-ci.org/>

### Einbindung Paketmanager und Grunt

Die Einbindung von Node.js, Bower und Grunt in den Continuous-Integration-Prozess funktioniert durch Kommandozeilenbefehle, siehe Abbildung 4.13, welche von Jenkins ausgeführt werden.

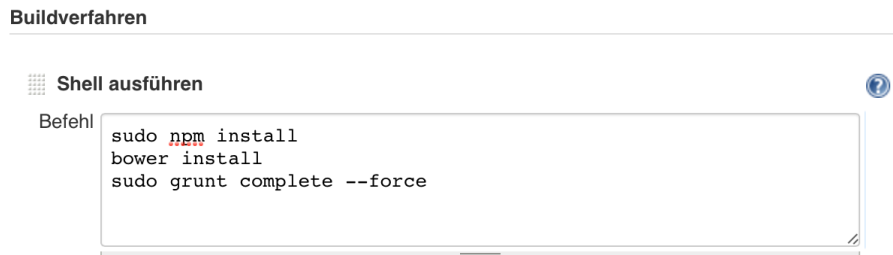


Abbildung 4.13: Integration von Paketmanagern und Grunt.

### Checkstyle

Um die Ergebnisse des ESLint-Checks in Jenkins anzeigen zu können, wird das Jenkins-Checkstyle-Plugin genutzt. Dazu muss in der Jenkins-Projekt-Konfiguration der Pfad zu den Ergebnissen angegeben werden. Anschließend wird in der Projektübersicht ein Trend-Graph, wie in Abbildung 4.14 zu sehen ist, mit der Anzahl an Checkstyle-Warnungen erstellt. Zusätzlich lassen sich diese Warnungen je nach Verzeichnis, Datei oder Warnungs-Typ aufschlüsseln, vergleiche Abbildung 4.15.

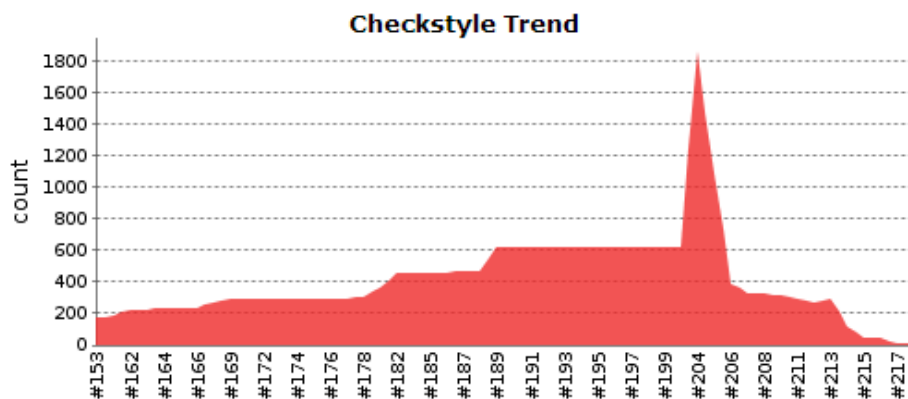


Abbildung 4.14: Trend-Graph der Checkstyle-Warnungen.

## Checkstyle Warnungen - Hohe Priorität

### Details

Verzeichnisse	Dateien	Warnungen	Details
Quellverzeichnis	Gesamt	Verteilung	
<a href="#">app</a>	2	<div><div></div></div>	
<a href="#">app/view</a>	7	<div><div></div></div>	
<a href="#">test</a>	5	<div><div></div></div>	
<a href="#">test/action</a>	5	<div><div></div></div>	
<a href="#">test/assertion</a>	13	<div><div></div></div>	
<a href="#">test/service</a>	1	<div><div></div></div>	
Gesamt	33		

Abbildung 4.15: Checkstyle-Warnungen – Hohe Priorität.

### Cobertura

Zur Anzeige der Zeilenabdeckung aus den Akzeptanztests wird das Cobertura Plugin gebraucht. Das Plugin erzeugt einen Trend-Graphen, in dem die Zeilenabdeckung des Codes untergliedert in Pakete, Dateien, Klassen, Methoden, Zeilen und Verzweigung dargestellt wird, siehe Abbildung 4.16. Über den Graphen kann bis in die einzelnen Dateien und den dazugehörigen Quellcode navigiert werden. Welche Teile des Codes durch die ausgeführten Tests bisher abgedeckt bzw. eben nicht abgedeckt sind, wird hier nachvollziehbar, wie in Abbildung 4.17 zu sehen ist.

#### Zusammenfassung der Testabdeckung nach Package

Name	Dateien	Klassen	Methoden	Zeilen	Verzweigungen
app	100% <div><div></div> 2/2</div>	100% <div><div></div> 2/2</div>	83% <div><div></div> 5/6</div>	92% <div><div></div> 34/37</div>	80% <div><div></div> 8/10</div>

#### Aufschlüsselung der Testabdeckung nach Datei

Name	Klassen	Methoden	Zeilen	Verzweigungen
<a href="#">Component.js</a>	100% <div><div></div> 1/1</div>	100% <div><div></div> 1/1</div>	100% <div><div></div> 15/15</div>	-
<a href="#">MyRouter.js</a>	100% <div><div></div> 1/1</div>	80% <div><div></div> 4/5</div>	86% <div><div></div> 19/22</div>	80% <div><div></div> 8/10</div>

Abbildung 4.16: Zeilenabdeckung im Paket *app*.

```

20 |         myNavBack : function(sRoute, mData) {
21 | 2         var oHistory = sap.ui.core.routing.History.getInstance();
22 | 2         var sPreviousHash = oHistory.getPreviousHash();
23 |         // The history contains a previous entry
24 | 2         if (sPreviousHash !== undefined) {
25 | 0             window.history.go(-1);
26 |         } else {
27 | 2         var bReplace = true;
28 |         // otherwise we go backwards with a forward history
29 | 2         this.navTo(sRoute, mData, bReplace);
30 |         }
31 |     },
32 |

```

Abbildung 4.17: Zeilenabdeckung im Quellcode.

### HTML Publisher

grunt-plato stellt die Ergebnisse der statischen Code-Analyse als HTML-Bericht zur Verfügung. Durch das Plugin wird in der Jenkins-Projektübersicht auf diesen Plato-Bericht verlinkt. Der Bericht zeigt Ergebnisse für das komplette Projekt oder auch für einzelne Dateien. In Abbildung 4.18 sieht man einen Report für die Datei Master3.controller.js.

### Timestampper

Fügt der Konsolenausgabe von Jenkins einen Zeitstempel hinzu.

### Workspace Cleanup

Ermöglicht das automatische Löschen des Arbeitsbereiches vor Ausführung des Builds. Da der komplette Prozess als Code abgebildet ist, wird permanent eine neue Build-Umgebung geschaffen.

### embeddable-build-status

Erstellt ein Icon wie in Abbildung 4.19 mit dem aktuellen Status des Builds zur Einbindung in Webseiten – In unserem Fall in die GitHub README.

build passing

Abbildung 4.19: Build-Status.



## view/Master3.controller.js

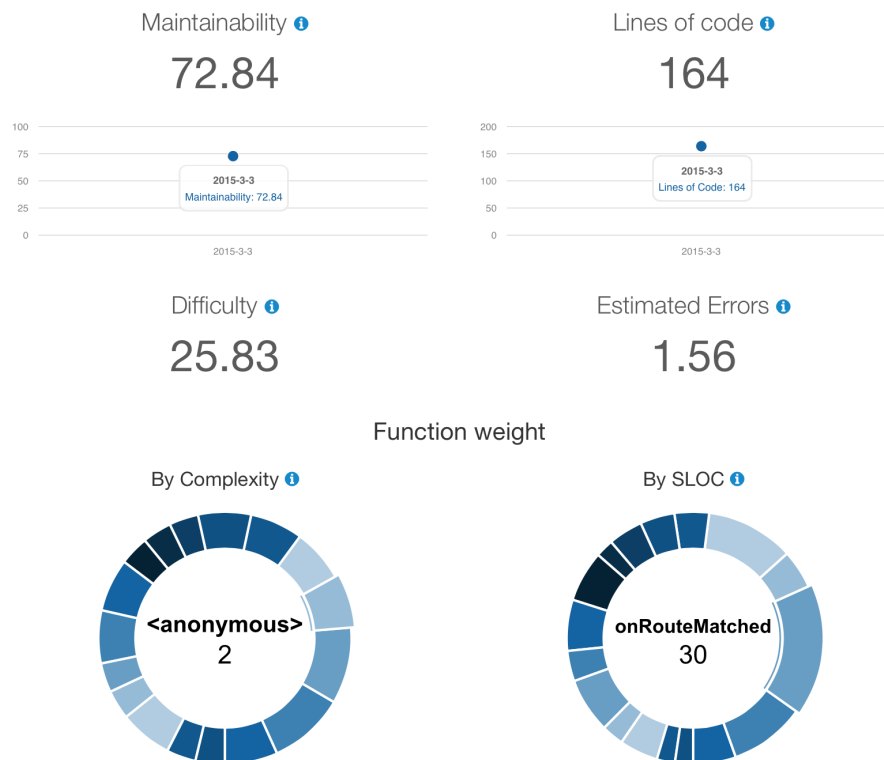


Abbildung 4.18: Plato-Bericht zu *Master3.controller.js*.

### 4.2.7 Zusammenfassung

Die Testumgebung setzt sich aus drei großen Teilschritten zusammen. Diese sind die Build-Auslöser, der eigentliche Build-Prozess sowie die Post-Build-Operationen.

Build-Auslöser sind Ereignisse, die Jenkins dazu veranlassen einen neuen Build zu erstellen und alle Tests durchzuführen, die für dieses Projekt definiert wurden. Dabei unterscheiden wir unter folgenden Build-Auslösern:

1. Der regelmäßige Build - jeden Tag um Mitternacht
2. Der Build bei Änderungen im GitHub - bei jedem Push
3. Der manuelle Start eines Buildvorganges

Nach dem gemeldeten Ereignis durch den Auslöser startet Jenkins den Build. Zu Beginn jedes Builds wird der Arbeitsbereich gelöscht, um eine neue Build-Umgebung zu schaffen und die aktuelle Version des Quellcodes aus dem

GitHub Repository herunterzuladen. Anschließend werden die benötigten Werkzeuge und Bibliotheken heruntergeladen und Grunt gestartet.

Im Build-Prozess werden schrittweise folgende Schritte ausgeführt:

1. Löschen des Arbeitsbereiches für einen sauberen Build
2. Code-Analyse durch ESLint und Plato
3. QUnit-Testausführung mit Karma inklusive Überprüfung der Zeilenabdeckung durch die QUnit bzw. OPA5-Tests
4. Kopieren der, für die App, notwendigen Dateien und Programmbibliotheken in den Target-Ordner
5. Komprimierung des JavaScript-Codes durch Uglify
6. Archivierung des Target-Ordners in eine ZIP-Datei
7. Hochladen der ZIP-Datei zu PhoneGap Build zum Kompilieren der Anwendung

Zum Abschluss werden aus den Ergebnissen der verschiedenen Tests Berichte erzeugt und über die Jenkins-Oberfläche bereitgestellt. Die fertig kompilierte Anwendung lässt sich nun über PhoneGap Build auf den Geräten installieren. Die Dateistruktur des Projektes sieht nun so aus:

- app/
- bower\_components/
- node\_modules/
- target/
- test/
- test-reports/
- .eslintrc
- .gitignore
- bower.json
- config.xml
- Gruntfile.js
- karma.conf.js
- package.json

## 4.3 UI5-App

### 4.3.1 Auswahl der Entwicklungsumgebung

#### Das bietet SAP

Die von der SAP vorgeschlagene Universal-Entwicklungsumgebung ist Eclipse. Sie gehört *noch* zur Firmenstrategie und wird mit einigen Plugins<sup>5</sup> einsatz-

---

<sup>5</sup><https://tools.hana.ondemand.com/>

bereit für die Entwicklung im SAP-Umfeld gemacht. Sie unterstützt unter anderem

- ABAP Development Tools for SAP NetWeaver
- SAP HANA Cloud Platform Tools
- Gateway (SAP Mobile Platform Tools)
- UI Development Toolkit for HTML5

Über die Transaktion `/UI5/UI5_REPOSITORY_LOAD` werden UI5-Apps als Business Server Pages (BSP) auf einem SAP Web Application Server bereitgestellt. Das ist die Standard-Methode für Fiori-Apps. Die Transaktion ist remotefähig und wird per HTTP-Aufruf in eine CI-Toolchain eingebunden. Die gleiche Deploymentfunktion in Eclipse stellt der ABAP Repository Team Provider bereit.

Nicht zwingend notwendig, aber nützlich für die Backend-Entwicklung, ist der grafische OData Modeler aus den SAP Mobile Platform Tools, siehe Abschnitt 4.1.1. Er erleichtert Kommunikation und Abstimmung zwischen den Entwicklern. Die generierten Metadaten können sowohl im Gateway, als auch in SAPUI5 genutzt werden, ohne dass der eigentliche Service schon implementiert sein muss.

Das UI Development Toolkit for HTML5 enthält für die Frontend-Entwicklung:

- App-Templates
- Test-Webserver
- Code-Vervollständigung
- XML-Syntaxprüfung

Ein schneller Einstieg in die UI5-Entwicklung ist mit diesen Tools möglich, sie sind jedoch keine Alleinstellungsmerkmale für Eclipse mit Plugin als integrierte Entwicklungsumgebung (IDE). Sie sind auch in anderen Umgebungen verfügbar. Vielmehr sind die Templates für größere Projekte oder Nutzung als Fiori App durch fehlende Modularisierung sogar ungeeignet.

Noch schneller gelingt der Start mit der neuen SAP Web IDE<sup>6</sup>. Die Umgebung basiert auf Eclipse Orion und ist nach dem Login direkt startbereit. Dort sind modulare Templates für Fiori- und Hybrid-Apps verfügbar, die den aktuellen Best Practices folgen. Zu den weiteren Funktionen zählen:

- Grafischer Layout-Editor
- Live-Vorschau
- Code-Vervollständigung
- HANA Cloud Platform und ABAP-Repositories
- Mock-Daten-Support

---

<sup>6</sup><http://scn.sap.com/docs/DOC-55465>

Die ersten Erfahrungen mit der Web IDE sind vielversprechend: Die Anwendung reagiert schnell und implementiert außerdem anpassbare Code-Checks per ESLint und git-Support. Die Entwicklung schreitet schnell voran und soll auf lange Sicht Eclipse als UI5-IDE ablösen.

Der Aufwand zur Entwicklung von Hybrid-Apps ist nicht verringert worden. Alle bekannten Build-Tools wie das Android SDK, Cordova, das Kapsel SDK u. a. müssen wie bisher lokal installiert und konfiguriert werden. Die Integration der Web IDE in externe Test- oder CI-Toolchains ist nicht möglich.

### Was benötigen wir?

Wichtig sind möglichst wenig Kontextwechsel während der Entwicklung und damit eine hochintegrierte Entwicklungsumgebung. Das Werkzeug soll in den Hintergrund rücken und TDD aktiv unterstützen. Wir wollen die gleiche Toolchain wie auf dem Integrationsserver nutzen. Konkret benötigen wir:

- Syntax-Hervorhebung der genutzten Sprache (JavaScript)
- Code-Vervollständigung der Frameworks (UI5)
- Integration von statischen Code-Analysen per ESLint
- QUnit-Testausführung per Karma
- Visualisierung der Code-Coverage
- Paketverwaltung per npm und Bower
- Grunt-Taskverwaltung

Eclipse beinhaltet im Auslieferungszustand nur wenig davon und die vorhandenen Funktionen wie Code-Vervollständigung in JavaScript-Projekten funktionieren nur selten. Manche der TDD-Funktionen ließen sich theoretisch per Plugin nachrüsten. Grunt als zentraler Task Runner aber z. B. nicht. Da die vorhandene SAP-Integration allein Eclipse als IDE nicht rechtfertigt, wählen wir eine alternative IDE mit JavaScript- und TDD-Fokus.

### Lösung: WebStorm

Gerade unter Web-Entwicklern genießt JetBrains WebStorm<sup>7</sup> einen hervorragenden Ruf: Die IDE ist speziell auf JavaScript-Projekte angepasst und bringt, neben zahlreichen Plugins, eine sinnvolle Vorkonfiguration mit. Schon aus früheren Projekten ist uns auch die Reaktionsfreude der zugrunde liegenden IntelliJ-Plattform bekannt. Wie können die bisher Eclipse-exklusiven Funktionen genutzt werden?

Die Syntax-Püfung von XML-Views ist über die entsprechenden XML-Definitionen (\*.xsd) verfügbar. Diese sind Bestandteil des UI5 Development Toolkits for Eclipse, aber auch separat auf GitHub erhältlich<sup>8</sup>. Das gleiche

---

<sup>7</sup><https://www.jetbrains.com/webstorm/>

<sup>8</sup><https://github.com/jbmurray/UI5-WebStorm-Files>

Repository enthält Bibliotheken, die sich für die UI5-Code-Vervollständigung in WebStorm nutzen lassen.

Das ABAP Team Repository kann nicht integriert werden. Einchecken als BSP aus WebStorm ist nur über den externen Aufruf einer Transaktion möglich. Für Fiori-Apps ist das verschmerzbar, da der CI-Server das finale Deployment übernimmt. Für hybride Apps spielt es ohnehin keine Rolle – Installationspakete werden auf anderen Wegen verteilt, z. B. über die App Stores oder eine Mobile Device Management (MDM)-Lösung wie SAP Afaria.

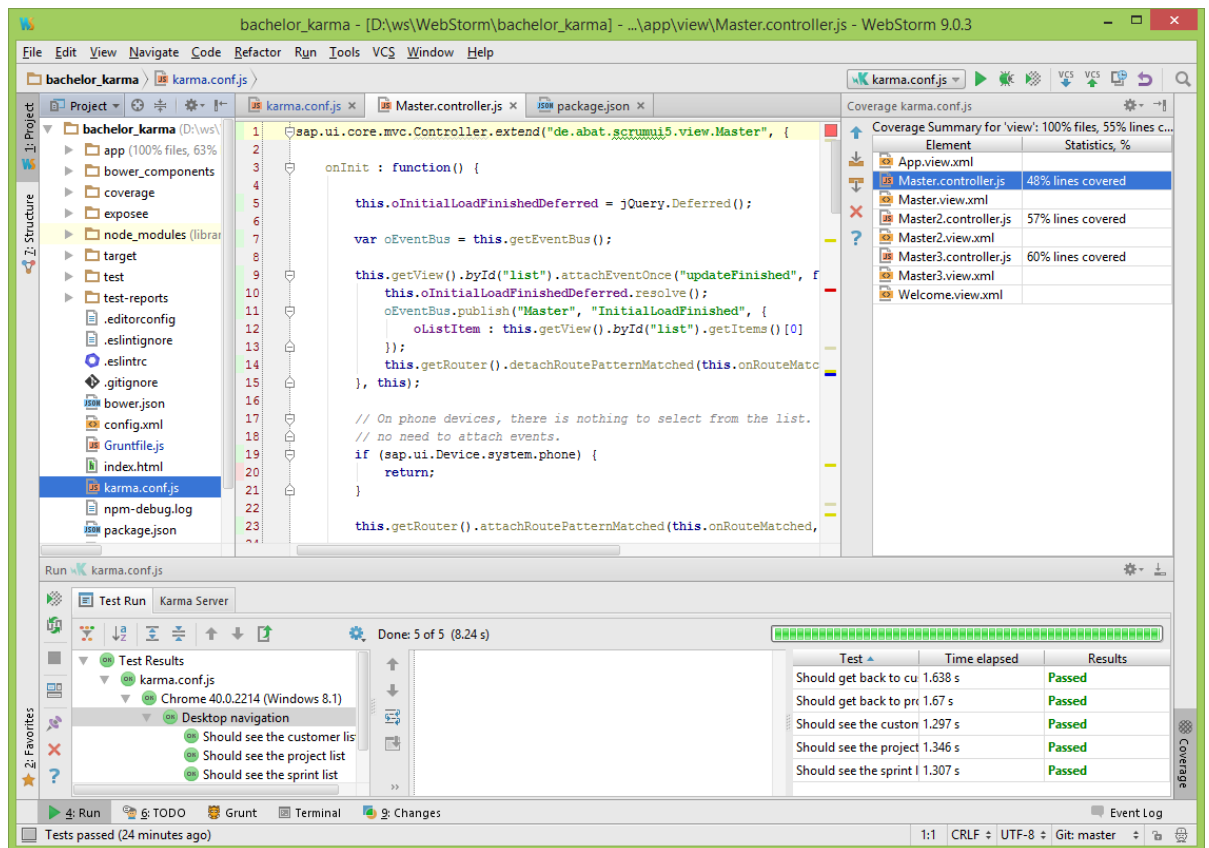


Abbildung 4.20: WebStorm mit TDD-Integration.

Webstorm integriert alle geforderten TDD-Funktionen. Abbildung 4.20 zeigt die Ausführung von Karma-Tests in der IDE. Test-Status und Code-Abdeckung werden direkt im Quellcode visualisiert. Grün markierte Zeilen wurden getestet, rote nicht. Um die testgetriebene Entwicklung in einer neuen WebStorm-Installation fortzusetzen sind folgende Schritte nötig:

1. Klonen des Projekt-Repositories
2. Installieren der Entwicklungs-Abhängigkeiten per *npm install*
3. Installieren der Frontend-Abhängigkeiten per *bower install*

Die Test-Konfiguration muss nicht angepasst werden – WebStorm nutzt die, bereits als Code vorliegende, CI-Toolchain mit ihren Grunt-Tasks automatisch. Die Toolchain wird nicht nur abgespielt, sondern kann auch mit IDE-Unterstützung bearbeitet werden. WebStorm hat eigene Paketverwaltungen für Node.js und Bower, die sich entsprechend nutzen lassen.

Auch viele der weiteren Konfigurationsdateien werden automatisch erkannt. Die statische Code-Analyse erfolgt parallel mithilfe der eigenen ESLint-Regeln. Automatische Code-Formatierung orientiert sich an der IDE-übergreifenden *.editorconfig*. Selbst die *.gitignore*-Konfiguration findet bei der Nachverfolgung lokaler Änderungen Verwendung.

Insgesamt wird die Entwicklung durch WebStorm aktiv unterstützt. Alles was automatisiert werden kann, wird es auch. Die Benutzererfahrung ist deutlich besser als unter Eclipse, wo vieles nur langsam oder gar nicht funktioniert. Gegenüber der SAP Web IDE ist die Integration und Weiterentwicklungsmöglichkeit der CI-Toolchain sehr gelungen.

### 4.3.2 UI5-App

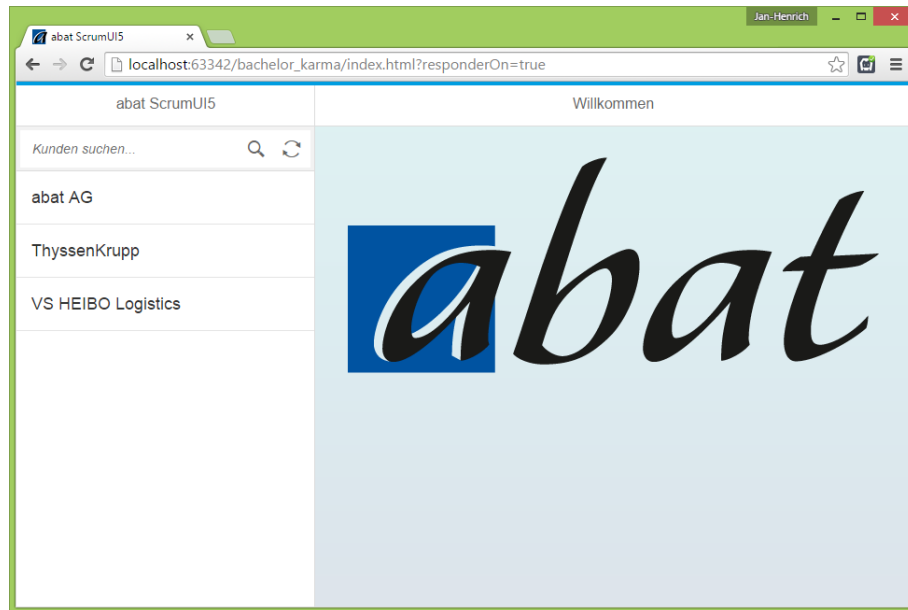
Abbildung 4.21 zeigt den Hauptbildschirm der entwickelten UI5-Scrum-App. Sie ist nach dem typischen Master-Detail-Konzept gestaltet: In der linken Master-View wird erst ein Kunde ausgewählt, dann ein Projekt und Sprint. Anschließend zeigt die große Detail-View rechts die entsprechenden Sprint-Daten. Die Ansicht ist responsive, wird bei kleineren Bildschirmen also automatisch angepasst.

Intern entspricht die App dem Model-View-Controller-Prinzip. Daraus ergibt sich folgende (gekürzte) Dateistruktur, die den aktuellen Best Practices entspricht:

- app/
  - i18n/
    - \* messageBundle.properties
    - \* messageBundle\_de\_DE.properties
  - view/
    - \* Master.view.xml
    - \* Master.controller.js
  - Component.js
  - index.html
  - MyRouter.js

### Kapselung

Als Startpunkt dient die *index.html*, sie ermöglicht die Ausführung der App im Browser. Metatags geben dem Browser Hinweise zur korrekten Darstellung.



**Abbildung 4.21:** UI5-App – Hauptbildschirm mit Mock-Daten.

Danach findet nur noch das Bootstrapping, also Laden, der UI5-Bibliotheken statt. Anschließend wird die App in einem Shell-Container gestartet.

Die eigentliche App ist unabhängig von einer HTML-Datei in der *Component.js* definiert. Durch diese Kapselung ist die App portabel. Eine weitere Ausführungsumgebung ist z. B. das Fiori Launchpad. In der Komponente sind allgemeine Informationen wie App-Name und Version gespeichert. Ebenso festgelegt ist die erste aufzurufende View.

### Routing

Daneben enthält *Component.js* die Routen-Konfiguration. Das Routing-Konzept ist Teil der Kapselung, da die Navigation innerhalb der App über einen eigenen Router (*MyRouter.js*) stattfindet. In Ausführungsumgebungen mit mehreren UI5-Apps werden Konflikte vermieden.

Weiterer Vorteil: Über bestimmte URLs ist immer der gleiche App-Zustand erreichbar. Wie von anderen Webanwendungen gewohnt, kann über den Parameter */Kunden(Kunde='ABAT')/Projekte* immer die Projektübersicht des Kunden, in diesem Fall abat, aufgerufen werden. Die entsprechende Seite lässt sich bequem als Lesezeichen speichern.

### Internationalisierung

Statische Texte sind in Resource Bundles wie *messageBundle\_de.properties* definiert. Sie enthalten Schlüssel/Wert-Paare mit der entsprechenden Über-

setzung in einer bestimmten Sprache und Landeskenntung. Englische Standardtexte sind grundsätzlich in *messageBundle.properties* gespeichert.

In einer XML-View wird statischer Text wie *i18n>masterTitle* automatisch mit dem Pendant aus dem i18n-Model ersetzt. Die aktuelle Sprache wird z. B. aus dem Browser, dem Betriebssystem oder einem URL-Parameter bestimmt. UI5 sucht automatisch das passende Resource Bundle.

## OData

Die Adresse des OData-Services ist ebenfalls in der *Component.js* festgelegt. Während der Initialisierung werden die Metadaten geladen und als globales App-Model festgelegt. Anschließend werden die Elemente einer Liste z. B. über *items='/Kunden'* entsprechend der Einträge im Pfad des OData-Services angezeigt. Bei einem Klick auf den Kunden, wird dieser als aktueller *BindingContext* gespeichert. Die Elemente der nachfolgenden Liste werden dann unter *gespeicherterKunde/Projekte* gesucht. An dieser Stelle greifen UI5 und die Navigationseigenschaften des OData-Services ineinander.

### 4.3.3 Akzeptanztests mit OPA5

OPA5 basiert auf QUnit und ermöglicht Akzeptanztests für UI5-Apps. Es erleichtert die Test-Programmierung durch spezielle Selektoren für UI5-Controls und die Verwaltung von Asynchronität.

Analog zu den Testfall-Szenarien im Unterabschnitt 3.2.2 werden Tests per *Given/When/Then* definiert. Die Formulierung aus den Testfällen lässt sich fast direkt in Code umsetzen, siehe Listing 12. Voraussetzung hierfür ist die vorherige Definition zahlreicher eigener *Actions* und *Assertions*.

Ein Beispiel ist die Assertion *iShouldSeeTheCustomerList()*. Diese wird aus *NavigationAssertions.js* nachgeladen und dem Framework über *extend-Config()* bekannt gemacht. Innerhalb der nachgeladenen Datei ruft sie generellere Funktionen wie *iClickOnAListItem(listName)* auf. Hierüber ist eine grundsätzliche Modularisierung der Tests möglich.

Nur für bestimmte Views nützliche Testfunktionen werden diesen über PageObjects zugeteilt und automatisch mitgeladen. Dies erleichtert die Strukturierung. Trotzdem wird das Testprojekt bei größerem Umfang schnell unübersichtlich. Die meisten Funktionen müssen selbst erstellt werden. Ein grafischer Editor, wie z. B. Jubula ihn mitbringt, wäre von großem Vorteil.

Die OPA5-Tests können über eine HTML-Datei gestartet werden. Diese dient nur als Container, analog zum Start der normalen App-Komponente. Dort werden sie wie klassische QUnit-Tests in einem iFrame ausgeführt. In unserer Toolchain genügt eine JavaScript-Datei wie Listing 12, die von Karma ausgeführt wird.

Die Implementierung des Mockservers ist bereits in Unterabschnitt 4.2.3 beschrieben. Er wird über den URL-Parameter *?responderOn=true* aktiviert.



Der Test läuft jetzt auch ohne Verbindung zum Backend.

```
sap.ui.require(
[
    'sap/ui/test/Opa5',
    'sap/ui/test/opaQunit',
    'de/abat/scrumui5/test/action/NavigationAction',
    'de/abat/scrumui5/test/arrangement/NavigationArrangement',
    'de/abat/scrumui5/test/assertion/NavigationAssertion'
],
function (Opa5, opaQunit, NavigationAction,
    NavigationArrangement, NavigationAssertion) {

    Opa5.extendConfig({
        actions: new NavigationAction(),
        arrangements: new NavigationArrangement(),
        assertions: new NavigationAssertion(),
        viewNamespace: "de.abat.scrumui5.view."
    });

    module("Desktop navigation");

    opaTest("Should see the customer list", function (Given, When, Then) {
        // Arrangements
        //Given.GivenIStartTheAppOnADesktopDevice();
        Given.iStartAComponent('de.abat.scrumui5');

        // Actions
        When.iLookAtTheScreen();

        // Assertions
        Then.iShouldSeeTheCustomerList().
            and.theCustomerListShouldHaveSomeEntries();
    });
});
```

Listing 12: OPA5-Test-Szenario 1

# Kapitel 5

## Schlussfolgerungen

### 5.1 Lessons Learned

#### 5.1.1 Architektur-Überlegungen

Die Möglichkeiten eine UI5-App zu veröffentlichen und zu nutzen sind vielfältig. Da die geschaffene Infrastruktur meist längerfristig genutzt wird, sollte sie nicht nur zur konkreten App, sondern zur gesamten Firmenstrategie passen. Besonders interessant sind folgende Aspekte:

1. Sind weitere UI5-Apps geplant?
2. Werden Fiori-Apps genutzt?
3. Existiert bereits eine MDM-Strategie?
4. Verteilung und Update der Apps
5. Aus welchem Netz erfolgen Zugriffe?
6. Welche Authentifizierungsserver werden genutzt?
7. Ist ein Offline-Zugriff nötig?
8. Sind native Gerätefunktionen gefordert?
9. Sollen eigene Plugins genutzt werden?
10. Umfang des Brandings/CI
11. Art und Standort der Backend-Systeme
12. Ist Cloud-Infrastruktur gewünscht?

Diesen Stichpunkten folgend, können die passende Architektur und zugehörige Produkte ausgewählt werden. Zusätzlich ergeben sich Art und Umfang der Teststrategie und besonders der Testwerkzeuge.

#### 5.1.2 OData-Service generieren oder selbst implementieren?

Bei geeigneten Schnittstellen ist die Generierung von OData-Services tatsächlich sehr gut geeignet, um schnell erste Ergebnisse zu liefern. So entsteht in kürzester Zeit ein funktionsfähiger Service mit grundlegenden Funktionen.

Für die Erweiterung und Optimierung von generierten Services ist dennoch ABAP-Programmierung nötig.

Ohne bereits vorhandene Schnittstellen oder bei komplexen Services empfiehlt es sich, direkt zur Selbstimplementierung des Services zu greifen, auch wenn der Initialaufwand etwas höher ist.

Die Entwicklung für das SAP-Backend lief, dank umfassender Dokumentation, gut. Der Aufwand steigt erheblich, sobald von Standard-Funktionen abgewichen wird und Customizing notwendig wird. Zum Beispiel das Filtern von Substrings bei Nichtbeachtung der Groß- und Kleinschreibung kann nicht über die bereitgestellten Standard-Methoden zum Auslesen von übergebenen OData-Parametern genutzt werden.

Insgesamt verschiebt sich der Aufwand hin zur Backendprogrammierung. Jede dort implementierte Funktion vereinfacht die Entwicklung der UI5-App. Ein gutes Beispiel sind die Navigationseigenschaften des OData-Modells. Sind sie korrekt implementiert, genügt im Frontend die korrekte Verlinkung, weitere Logik muss nicht implementiert werden.

### 5.1.3 Wo ist die aktivste UI5-Community?

Mit der Veröffentlichung des UI5-Codes auf GitHub verschieben sich auch die Anlaufstellen für Fehlermeldungen und Hilfestellungen. Während der Kontakt zu den Framework-Entwicklern vor OpenUI5 kaum möglich war, können Fehlermeldungen oder Hinweise nun direkt über das GitHub Issue Management eingestellt werden. Zum Beispiel im Fall des fehlerhaften *karma-openui5* Plugins hat sich so ein interessanter Dialog ergeben. Der veröffentlichte Code gewährt außerdem Einblicke in zukünftige Best Practices in Form neuer Beispiel-Apps und den zugehörigen Testfällen.

Das SAP Community Network (SCN)<sup>1</sup> ist weiterhin die wertvollste Resource für Blogs und umfangreiche Tutorials. Auch komplexe Themen, wie das Kapsel SDK, werden mit Beispielen eingeführt und erleichtern den Entwicklungsstart ungemein. Daneben könnte StackOverflow als renommiertes Programmier-Forum nützlich werden. Die Zahl der Posts unter den Tags *openui5* und *sapui5*<sup>2</sup> nimmt ständig zu.

### 5.1.4 Infrastruktur als Code

Das Prinzip *Infrastructure as Code* ist nicht nur essenziell für die verteilte testgetriebene Entwicklung, es wird durch den hauptsächlichlichen Einsatz von JavaScript auch erheblich gefördert. Die gesamte Toolchain von Task Runner über Paketmanager und Testtreiber wird automatisiert aus den Konfigurationsdateien wiederhergestellt.

---

<sup>1</sup><http://scn.sap.com/community/developer-center/front-end>

<sup>2</sup><http://stackoverflow.com/questions/tagged/sapui5>

Selbst beim erstmaligen Herunterladen aus der Versionskontrolle ist das Projekt in IDE oder CI-Server mit zwei Befehlen einsatzbereit. Auf dem CI-Server machen wir uns diese Eigenschaft zunutze, indem das Projektverzeichnis vor jedem Build neu aufgebaut wird. Mögliche Nennwirkungen werden vermieden und die Lauffähigkeit der aktuell eingechekten Infrastruktur ist sichergestellt.

### 5.1.5 Alternative IDE prüfen

Auch wenn SAP Eclipse als Standard-Entwicklungsumgebung für fast alle Produkte präsentiert – Für eine UI5-Entwicklung auf JavaScript-Basis gibt es interessante Alternativen mit hervorragender Testintegration.

Eclipse ist keine beliebte Webentwicklungsumgebung, laut SCN<sup>3</sup> auch nicht innerhalb der SAP. Hinzu kommt die neue SAP Web IDE als Konkurrenz und langfristige Ablösung einer lokalen Eclipse-Installation. Ihr erster Eindruck ist gut: Die Umgebung ist sofort einsatzbereit, reagiert schnell auf Eingaben, bringt aktuelle, Fiori-kompatible Templates mit und integriert statische Code-Analysen. Einzig die Entwicklung von hybriden Apps wurde nicht vereinfacht, die CI-Integration ist noch gar nicht möglich.

Je nach eigenen Vorlieben und Projektvorgaben kann WebStorm eine gute Alternative zu Eclipse und der Web IDE sein. Genauso wie Letztere ist WebStorm schnell einsatzbereit und arbeitet zügig. Zusätzlich können auch Unit- und Akzeptanztests ausgeführt und Ergebnisse visualisiert werden. Paketmanager sind integriert und auf Wunsch läuft die gesamte CI-Toolchain lokal in der IDE. Gerade in Hinblick auf TDD ist die Benutzererfahrung mit WebStorm wesentlich besser.

### 5.1.6 Testmethoden kombinieren

TDD stellt sicher, dass der Code richtig geschrieben wurde, während BDD dafür sorgt, dass es auch der richtige Code ist. Ergänzend haben sich explorative Tests als nützlich erwiesen.

Diese sollten allerdings nicht nur spontan, sondern in Form bestimmter Szenarien oder Aufträge erfolgen. Ein Auftrag könnte lauten: *Überprüfe die Navigation in der App*. Sollten hierbei neue Fehlerwirkungen auftreten, schreiben wir einen entsprechenden Testfall und korrigieren den Code erst danach. Durch automatische Regressionstests wird der Fehler zukünftig vermieden.

---

<sup>3</sup><http://scn.sap.com/community/developer-center/front-end/blog/2014/09/22/configuring-jetbrains-webstorm-for-ui5-development>

### 5.1.7 Tests gezielt einsetzen – Nicht doppelt testen

Durch die Offenlegung des UI5-Quellcodes auf GitHub ist die testorientierte Entwicklung seitens SAP deutlich geworden. Das gesamte Framework ist durch zahlreiche Unit- und Akzeptanztests geprüft. Gleiches gilt für die zugrundeliegenden Basis-Frameworks wie jQuery und QUnit.

Probehalber haben wir die UI5-Tests ausgeführt – Der mehr als 15 Minuten dauernde Testlauf endete mit mehreren Fehlermeldungen und Warnungen. Was bedeutet das für unser Projekt?

1. Auch UI5 ist natürlich nicht fehlerfrei.
2. Keiner der fehlgeschlagenen Testfälle ist für uns relevant.
3. Teilweise sind noch nicht implementierte Funktionen betroffen.
4. Der gesamte Testlauf dauert sehr lange.
5. Die Ergebnisanalyse ist extrem aufwändig.

Wir konzentrieren uns daher auf Tests eigener Funktionen.

### 5.1.8 Statische Analyse differenziert betrachten

Die Bereitstellung offizieller UI5-ESLint-Regeln zur statischen Codeanalyse ist der richtige Schritt, um die Code-Qualität zu verbessern und den gleichen Code-Stil projektübergreifend durchzusetzen.

Das aktuelle UI5-Framework genügt diesen selbst gesetzten Anforderungen jedoch nicht: Eine Überprüfung mit ESLint zeigt so viele Warnungen, dass mit einem kompletten Refactoring auf absehbare Zeit nicht zu rechnen ist. Daraus folgern wir für unsere Projekt:

1. Die genutzten Frameworks von der statischen Analyse ausschließen.
2. Für zukünftige Kompatibilität eigenen Code regelkonform gestalten.

Bei der Übernahme einer bereits vorhandenen Codebasis ohne fest definierte Regeln kann kaum ein bestimmtes Regelset durchgesetzt werden. Die Änderungen sind oft viel zu umfangreich und kommen einer Neuprogrammierung nahe. Stattdessen empfiehlt sich die Suche nach einem Stil der möglichst genau dem des vorhandenen Codes entspricht und eine Anpassung ermöglicht.

Im Notfall ist die Gewichtung der Fehler per Regel herunterzusetzen oder bestimmte Optionen ganz zu deaktivieren. Neue Fehler gehen sonst im Protokoll unter. Eine statische Analyse die standardmäßig viele Warnungen ausgibt, stumpft ab und bringt keinen Mehrwert.

### 5.1.9 Code Coverage ist nicht alles

Die Code Coverage ist zwar leicht zu messen, aber keine geeignete Metrik um das erfolgreiche Testende festzustellen. Stattdessen liefert sie Hinweise auf ungenutzten Code, der durch ein Refactoring entfernt werden kann.

Für die Wartbarkeit haben sich die Plato-Metriken als interessant herausgestellt. Sie analysieren die Komplexität des Codes und decken Modularisierungspotential auf.

## 5.2 Ausblick

Es bleibt ein breit gefächertes Integrationspotenzial. Während die Arbeit eine Einführung in TDD und CI mit UI5 darstellt, bleiben weitere Funktionen, App-Architekturen und Testwerkzeuge die, je nach Projektart und Größe, weiter untersucht werden sollten.

### 5.2.1 *Echte* Unit-Tests

Wie bereits festgestellt, handelt es sich bei unserer beschriebenen Vorgehensweise aktuell nicht um klassisches TDD mit Unit-Tests – Vielmehr nutzen wir Akzeptanztest.

Dieses Vorgehen hat sich bei der kleinen Projektgröße und dem geringen Umfang der App bewährt. Sobald eigene Funktionen in der App implementiert werden, zum Beispiel Formatter, werden Unit-Tests benötigt.

Mit der aktuellen Architektur haben wir alle Grundlagen dafür gelegt. QUnit als Testframework ist bereits UI5-Bestandteil und wird von unserem Testtreiber unterstützt. Auch die übrige CI-Toolchain ist auf den Einsatz vorbereitet.

### 5.2.2 Native Funktionen testen

Zukünftige App-Versionen werden native Geräte-Funktionen nutzen. Auch diese müssen in die Teststrategie einbezogen und automatisch geprüft werden. Aktuell gibt es für einen Test ohne Mobilgeräte oder Emulatoren folgende Möglichkeiten:

- Browser-Plugins<sup>4</sup>
- Mock-Frameworks<sup>5</sup>
- Cordova-Browser-Plattform<sup>6</sup>

Cordova-Browser-Plugins und -Mock-Frameworks arbeiten nach dem gleichen Schema: Sie bieten einen Mock der cordova.js-API an. Dieser wird entweder direkt in den Testressourcen eingebunden oder in eine laufende Browsersitzung injiziert. API-Aufrufe liefern vordefinierte Werte oder zeigen einen Dialog zur Eingabe der Testdaten.

Die vielversprechendste Möglichkeit ist die neue, offizielle Cordova-Plattform *Browser*. Sie ist seit 2014 als Beta-Version verfügbar und ermöglicht

---

<sup>4</sup><https://github.com/pbernasconi/chrome-cordova>

<sup>5</sup><https://github.com/ecofic/ngCordovaMocks>

<sup>6</sup><https://github.com/apache/cordova-browser>

einen problemlosen Betrieb von Hybrid-Apps in Desktop-Umgebungen. Das *deviceready*-Event wird automatisch abgesetzt und Standard-APIs wie Kamera und GPS stehen zur Verfügung. Aktuell ist die Einrichtung allerdings noch aufwändig.

### 5.2.3 Tiefere SMP-Integration

Aktuell nutzen wir die SAP Mobile Platform in Form der HANA Cloud Platform mobile services: Sie stellt unseren OData-Service außerhalb des Intranets bereit. Zusammen mit den Cordova-Plugins des Kapsel-SDKs bietet es noch mehr Möglichkeiten<sup>7</sup>:

- Single Sign-on
- OData-Offline-Funktionen
- Push-Nachrichten
- App-Updates
- Statistiken

Gerade Single-sign on ist für eine zukünftige Mobil-Strategie auf UI5-Basis Voraussetzung, um mehrere Apps komfortabel nutzbar zu machen. Statt des SAP-Kontos als Fiori-Standard, können über die Mobile Platform auch andere Authentifizierungsstrukturen eingebunden werden, zum Beispiel ein Active Directory.

Offline-Funktionen stellen für die App-Entwicklung eine große Herausforderung dar. Die Eigenimplementierung von Synchronisierungsfunktionen ist komplex und damit fehleranfällig. Hier unterstützt die Kombination aus SAP Mobile Platform und Kapsel-Plugin – Sie erweitert auch vorhandene OData-Services und UI5-Apps unter iOS und Android um Offline-Funktionen. Gleiches gilt für Push-Nachrichten.

Automatische App-Updates können für bereits installierte Apps über die Mobile Platform veröffentlicht werden. Dies kann allerdings nur eine Ergänzung zu einem vorhandenen Mobile Device Management wie Afaria sein. Nur darüber gelingt die Erstverteilung.

### 5.2.4 Fiori Deployment

Durch die strikte Einhaltung der UI5-Best-Practices eignet sich die App auch zur Veröffentlichung über Fiori: Die Modularisierung ermöglicht einen Betrieb der App allein über das Laden der *Component.js*. Ob dies über eine HTML-Datei oder im Fiori-Launchpad erfolgt, spielt keine Rolle. Ressourcen und Navigations-Routen der App sind gekapselt und garantieren einen konfliktfreien Betrieb als Fiori-App.

In diesem Zusammenhang muss die parallele Veröffentlichung über das Fiori-Launchpad als BSP weiter geprüft werden. Ein manueller Upload der

---

<sup>7</sup><http://scn.sap.com/docs/DOC-49592>

App widerspricht dem CI-Gedanken. Der ABAP Team Provider (Eclipse) oder das Aufrufen der Transaktion `/UI5/UI5_REPOSITORY_LOAD` scheiden als Werkzeuge aus. Da der Baustein remotefähig ist, könnte eine CI-Integration über einen RFC erfolgen.

### 5.2.5 Continuous Delivery mit Afaria

Entwicklungs- und Testprozess sind automatisiert, doch wie kommt die Anwendung danach auf die mobilen Geräte der Mitarbeiter?

SAP bietet mit Mobile Secure eine Enterprise Mobility Management (EMM)-Lösung in der Cloud. Diese stellt durch die Integration von SAP Afaria unter anderem MDM zur Verwaltung von mobilen Geräten im Unternehmen bereit. Außerdem werden per Mobile Application Management (MAM) Anwendungen im firmeninternen App Store über den *SAP Mobile Place* veröffentlicht.

Durch Schnittstellen lassen sich die Installationspakete der Apps direkt aus der CI-Toolchain im *SAP Mobile Place* bereitstellen. Die Mitarbeiter oder auch Geschäftspartner können Anwendungen anschließend aus dem Anwendungskatalog auswählen und installieren. Auf Geräten, die durch das MDM verwaltet werden, ist außerdem eine automatische Installation und Aktualisierung der Anwendungen möglich. Alternativ ist SAP Afaria auch als Stand-Alone-Lösung einsetzbar<sup>8</sup>.

### 5.2.6 eCATT-Integration

Das Testen der Funktionsbausteine, sowie der OData-Services wurde in diesem Projekt aus Zeitgründen nicht automatisiert. Dies würde sich über das extended Computer Aided Test Tool (eCATT) realisieren lassen.

Die Funktionen zum Testen von Funktionsbausteinen ist standardmäßig in eCATT integriert. Automatisierte Tests von OData-Services benötigen zusätzlich die *Gateway Test APIs*, welche im SCN veröffentlicht wurden<sup>9</sup>.

## 5.3 Bewertung und Herausforderungen

Ziel der Arbeit war ein bereits vorhandenes ABAP-Altsystem mit einer neuen UI5-App benutzerfreundlicher zu gestalten. Wir haben dabei die neue SAP-Mobile-Strategie eingeschätzt und die entsprechenden Produkte kennengelernt. Seit der Sybase-Übernahme durch SAP wurde die ehemalige Sybase Unwired Platform zur SAP Mobile Platform weiterentwickelt – Mittlerweile auch als Mietangebot in Form der Hana Cloud Platform mobile services. Überhaupt existiert nun ein großes Ökosystem zum Thema Mobile Devices

<sup>8</sup><http://scn.sap.com/docs/DOC-62378>

<sup>9</sup><http://scn.sap.com/community/gateway/blog/2013/11/27/ecatt-based-test-automation-for-odata-services-available>



das z. B. um MDM-Lösungen erweitert wurde. In dieser Arbeit haben wir die gängigste Kombination aus universeller UI5-App und SAP Gateway umgesetzt.

Gegenüber älteren Frameworks wie jQuery Mobile ist UI5 ein vollwertiges, offenes Businessframework. Übliche Konzepte wie Routing, Internationalisierung und Modularisierung heben es auf eine Stufe mit großen Alternativen wie Sencha Ext JS. Die typische Schnellebigkeit anderer JavaScript-Frameworks lässt sich auch hier beobachten: Die eigenen Codestyle-Vorgaben werden noch nicht eingehalten und sind eher Ziel, als aktueller Standard. Die interne App-Struktur hat mehrere grundlegende Änderungen durchlaufen, die Best Practices werden das nächste Mal mit dem kommenden Release 1.28 umfassend geändert.

Das Konzept des SAP Gateways als Schnittstelle zwischen Backend und Frontend überzeugt. Die Schwierigkeit bei der Erstellung von OData-Services steigt mit der Komplexität. Als Frontend-Entwickler sind tatsächlich keine SAP-Kenntnisse notwendig. OData als Protokoll ist einfach und schnell zu verstehen. Besonders die Navigationseigenschaften zwischen den einzelnen Entitäten vereinfachen die Implementierung der Navigation in der Anwendung.

UI5 selbst wird mit großer Rücksicht auf Testen entwickelt, die umfangreiche Testsuite verdeutlicht dies. Schwieriger hat sich die Integration in eine CI-Toolchain erwiesen. Das Karma-Plugin ist noch fehlerhaft und benötigt einen Workaround. Best Practices zum Einsatz von Akzeptanztests mit OPA5 gibt es noch nicht, wir haben sie selbst erarbeitet. Im Vergleich zu Tools wie Jubula ist die Modularisierung aufwändiger, in großen Projekten könnte der Überblick leiden. Die fehlenden Erfahrungen sind kein UI5-spezifisches Problem: Die Kombination aus JavaScript, testgetriebener Entwicklung und Geschäftsanwendungen verändert sich schnell und wird seit einiger Zeit verstärkt mit eigener Literatur wie [3], [16] und [12] behandelt.

Insgesamt haben wir eine effiziente CI-Toolchain auf OpenSource-Basis etabliert, die wir auch in zukünftigen Projekten nutzen werden. Der testgetriebene Ansatz war aufwändig. Durch die Schaffung universeller Templates für die Akzeptanztests und der gesamten Projektstruktur werden kommende Entwicklungen deutlich weniger Aufwand erfordern. Wir werden diesen Ansatz weiterführen, da er automatische Regressionstests auf vielen verschiedenen Plattformen parallel erlaubt.

# Quellenverzeichnis

## Literatur

- [1] Miroslav Antolovic. *Einführung in SAPUI5*. Bonn: Galileo Press, März 2014 (siehe S. 12).
- [2] Carsten Bönner u. a. *OData und SAP Gateway*. Bonn: Galileo Press, Juni 2014 (siehe S. 12–16, 20, 26, 27, 31, 32, 37, 38).
- [3] Yakov Fain u. a. *Enterprise Web Development. Building HTML5 Applications: From Desktop to Mobile*. Sebastopol: O'Reilly Media, Juli 2014 (siehe S. 9, 67).
- [4] Tilo Linz und Andreas Spillner. *Basiswissen Softwaretest*. 4. Aufl. Heidelberg: dpunkt.verlag GmbH, Mai 2010 (siehe S. 24).
- [5] Damir Majer. *Unit-Tests mit ABAP Unit*. Heidelberg: dpunkt.verlag GmbH, Nov. 2008 (siehe S. 12).
- [6] Arno Mielke und Gerhard Henig. „SAP Mobile Überblick“. In: *DSAG-Technologietage 2015*. (Mannheim). Hrsg. von Hans-Achim Quitmann. Walldorf: DSAG Dienstleistungen GmbH, Feb. 2015, S. 19 (siehe S. 9).
- [7] Macario Polo, Mario Piattini und Francisco Ruiz. *Advances in Software Maintenance Management: Technologies and Solutions*. LaMancha: Igi Global, Nov. 2002 (siehe S. 4).
- [8] Golo Roden. „Werkzeuge für die Webentwicklung“. In: *ix* 10 (2015), S. 84–86 (siehe S. 5).
- [9] Richard Seidl, Manfred Baumgartner und Thomas Bucsics. *Basiswissen Testautomatisierung. Konzepte, Methoden und Techniken*. Heidelberg: dpunkt.verlag GmbH, März 2012 (siehe S. 5).
- [10] Andreas Spillner und Karin Vosseberg. „Codeüberdeckung garantiert keine Qualitätssoftware“. In: *Testing Experience. Das Magazin für Tester, Entwickler und Manager* (Apr. 2014), S. 39–43 (siehe S. 6).
- [11] Andreas Spillner u. a. *Praxiswissen Softwaretest – Testmanagement. Aus- und Weiterbildung zum Certified Tester – Advanced Level nach ISTQB-Standard*. 4. überarb. u. erw. Aufl. Heidelberg: dpunkt.verlag GmbH, Mai 2014 (siehe S. 20).

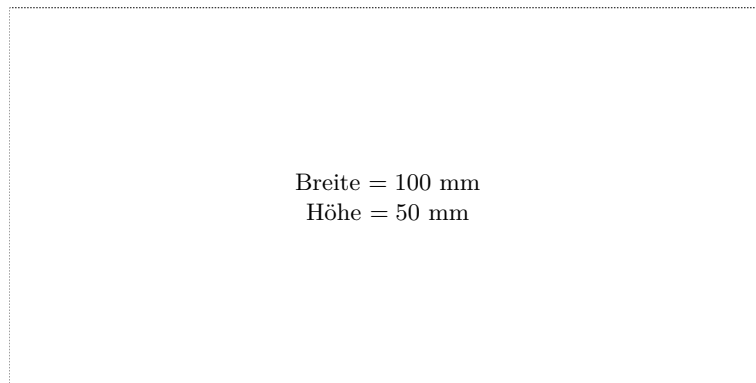
- [12] Sebastian Springer. *Testgetriebene Entwicklung mit JavaScript. Ein Handbuch für den professionellen Programmierer*. Heidelberg: dpunkt.verlag GmbH, Feb. 2015 (siehe S. 4, 67).
- [13] The Institute of Electrical and Electronics Engineers, Inc. *IEEE 829 Standard for Software Test Documentation*. Version 829-2008. 2008. URL: <http://standards.ieee.org/findstds/standard/829-2008.html> (siehe S. 20).
- [14] Matthias Thiebes. „Implementierung eines Kundenportals auf Basis SAPUI5“. In: *DSAG-Technologietage 2015*. (Leipzig). Hrsg. von Dr. Marco Lenck. Walldorf: DSAG Dienstleistungs GmbH, Okt. 2014, S. 16 (siehe S. 9).
- [15] Ray Valdes u. a. *Magic Quadrant for Mobile Application Development Platforms*. Techn. Ber. G00261830. Stamford, CT: Gartner, Inc., Sep. 2014. URL: <https://www.gartner.com/doc/2835919/magic-quadrant-mobile-application-development> (siehe S. 9).
- [16] Gunnar Wrobel. *JavaScript Tools. Besserer Code durch eine professionelle Programmierumgebung*. München: Open Source Press GmbH, Feb. 2015 (siehe S. 45, 67).

## Online-Quellen

- [17] Marc Chan. *Installing and Configuring SAP NetWeaver Gateway 2.0*. Version 2.0. Sep. 2011. URL: <http://scn.sap.com/docs/DOC-16328> (besucht am 12.10.2014) (siehe S. 12).
- [18] Markus Gärtner. *Qualitätssicherung ausgewogen*. Jan. 2013. URL: <http://heise.de/-1792246> (besucht am 13.02.2015) (siehe S. 3).
- [19] SAP SE. *Development Conventions and Guidelines*. Version 68f22b0. Nov. 2014. URL: <https://github.com/SAP/openui5/blob/master/docs/guidelines.md> (besucht am 23.11.2014) (siehe S. 12, 20).
- [20] SAP SE. *ESLint Code Checks*. Version 7cc4113. März 2015. URL: <https://github.com/SAP/openui5/blob/master/docs/eslint.md> (besucht am 03.03.2015) (siehe S. 45).
- [21] SAP SE. *Introduction to SAP HANA Cloud Platform. Advanced Features – SAP Mobile Platform Cloud Version*. Juni 2014. URL: <https://open.sap.com/course/hanacloud1-1> (besucht am 18.03.2015) (siehe S. 19).
- [22] SAP SE. *OData Query Options*. Version 2.0. März 2015. URL: [http://help.sap.com/saphelp\\_gateway20sp10/helpdata/en/71/d07b52a3566f54e10000000a44176d/content.htm](http://help.sap.com/saphelp_gateway20sp10/helpdata/en/71/d07b52a3566f54e10000000a44176d/content.htm) (besucht am 13.03.2015) (siehe S. 34).

# Messbox zur Druckkontrolle

— Druckgröße kontrollieren! —



— Diese Seite nach dem Druck entfernen! —