CRC Cards

Software-Entwicklungsprozesse in Seaside und JavaScript
Softwaretechnik I, SS2012

Kai Fabian, Dominik Moritz, Matthias Springer, Malte Swart

Hasso-Plattner-Institut

14. Juni 2012

Überblick

Projekt

Architektur

Entwicklungsprozess

Auswertung

CRC-Karten

CRC: Class, Responsibility, Collaboration

MoneyTransfer Superclass: Queuable

Subclasses

- StandingOrder
- InternationalTransfer

Collaborators

- BankAccount
- Transaction

Responsbilities

- Check limits
- Enqueue
- Ensure atomicity

Abbildung: Eine einfache CRC-Karte

Handout only: CRC-Karten

- Hilfsmittel für objektorientiertes Design
- Bieten einen schnellen Überblick über die Beschaffenheit einer Klasse:

Class	eindeutiger Klassenname
Responsibility	Liste aller Tätigkeiten/Aktionen, für deren Ausführung die Klasse verantwortlich ist
Collaboration	Liste aller Klassen, mit denen die beschriebene Klasse zusammenarbeiten muss, um ihre Verantwortlichkeiten zu erfüllen

Rollen im System

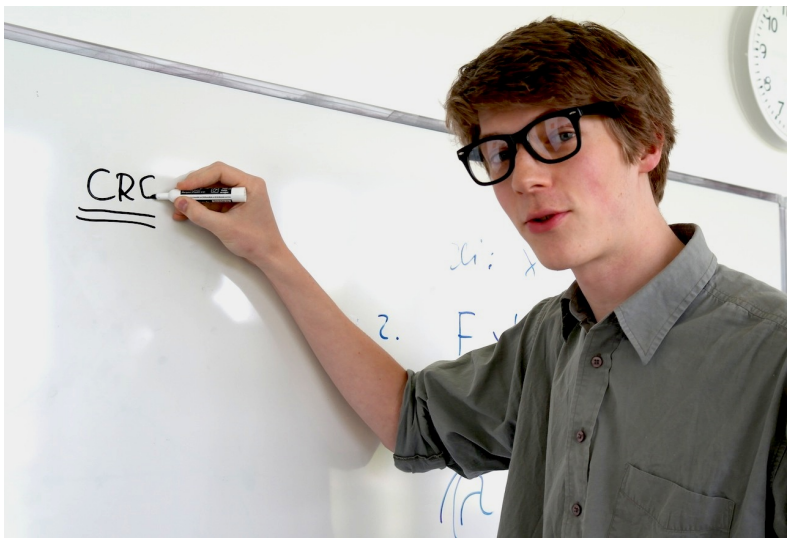


Abbildung: Projektleiter

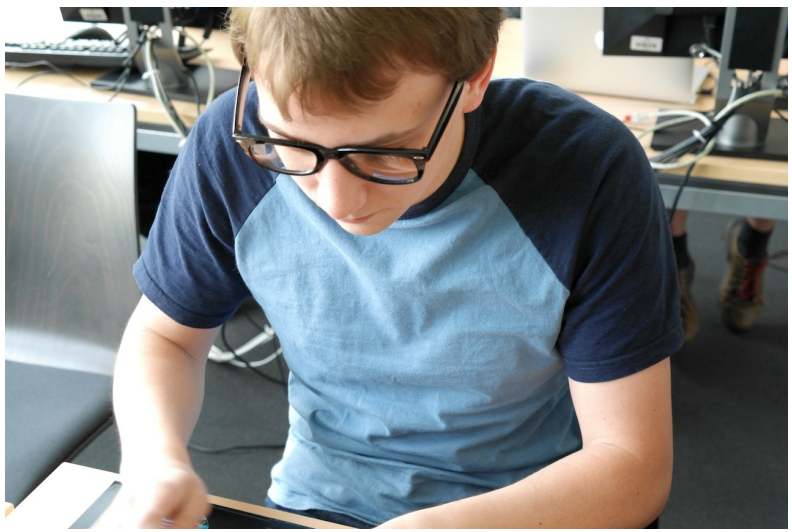


Abbildung: Entwickler

Nichtfunktionale Anforderungen (TEX-cloud)

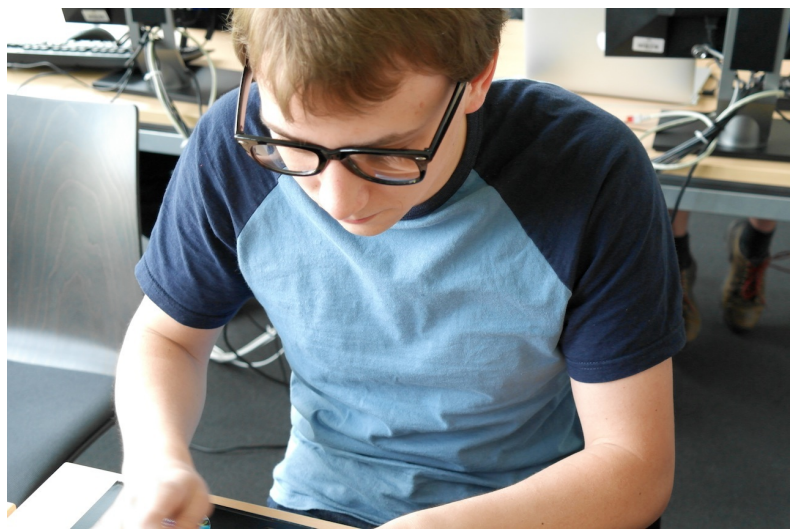
- Wenige, dafür ausgereifte Funktionalitäten
- Seaside/Smalltalk als Grundlage
- **Benutzbarkeit** und Skalierbarkeit
  - Benutzung des CRC-Editors ohne Maus
  - Bis zu 100 CRC-Karten pro Projekt
- Funktioniert in aktuellem Firefox oder Webkit mit aktiviertem JavaScript
- Keine Anforderungen bzgl.
  - Internationalisierung
  - Look-and-feel
  - Besondere Sicherheit

Funktionale Anforderungen (TEX-cloud)

- HPI-OpenID-Login
- CRC-Karten-Management mit Workspaces
- Projekte/Teams administrieren
- User Stories verwalten
- CodeMapping für Smalltalk

User Story: Login

" Als **Entwickler** möchte ich mich mit meiner HPI-OpenID auf der Seite einloggen. "



User Story: CRC-Karte erstellen

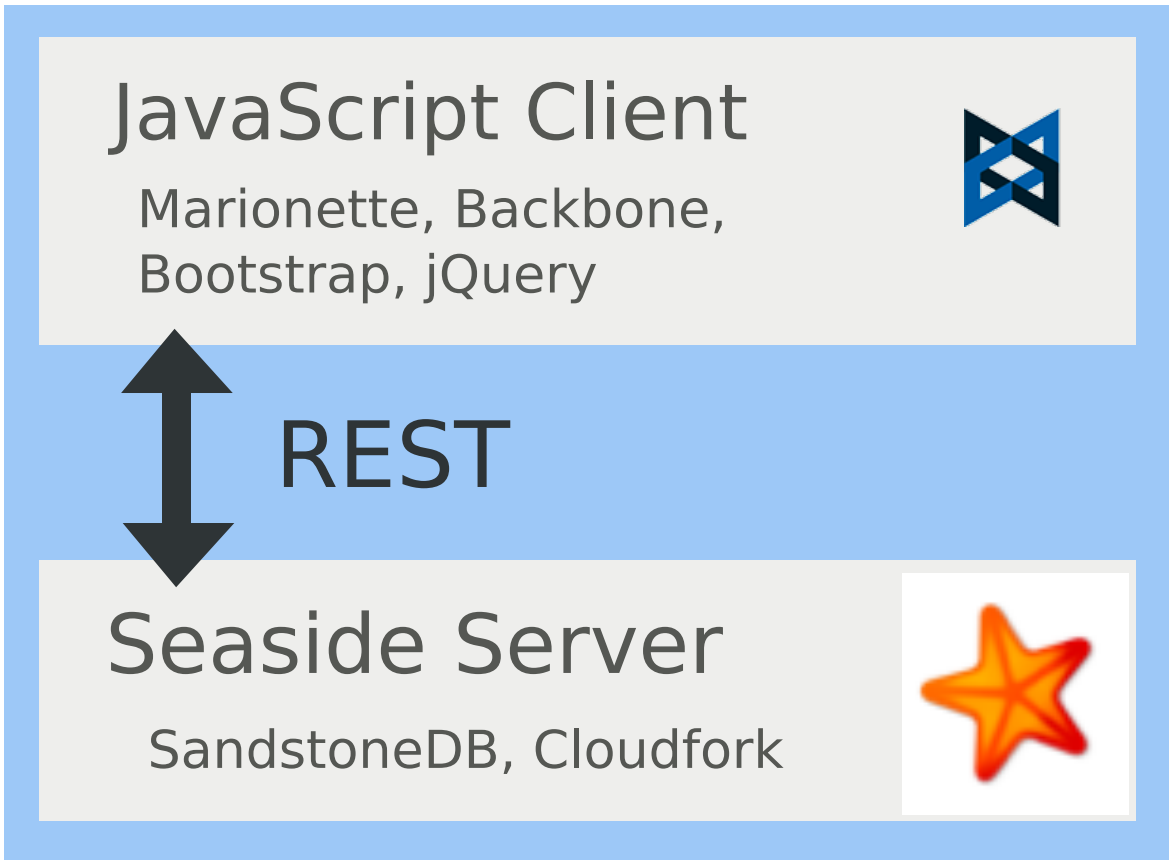
" Als **Entwickler** möchte ich CRC-Karten auf einem Workspace erstellen. "

User Story: CRC-Karte bearbeiten

" Als **Entwickler** möchte ich bestehende CRC-Karten bearbeiten. "

Überblick

Grobarchitektur



Handout only: Grobarchitektur

- Bilinguale Entwicklung: Backend in Smalltalk, Frontend in JavaScript
- Frontend
  - Stark interaktionsgetrieben
  - Look-and-Feel einer Desktop-Anwendung
  - Geringe Reaktionszeiten
- Datengetriebenes Backend
  - Zusätzliche Validierung (Sicherheit)
  - Persistieren der Daten
  - OpenID-Login (später mehr dazu)

Verantwortlichkeiten

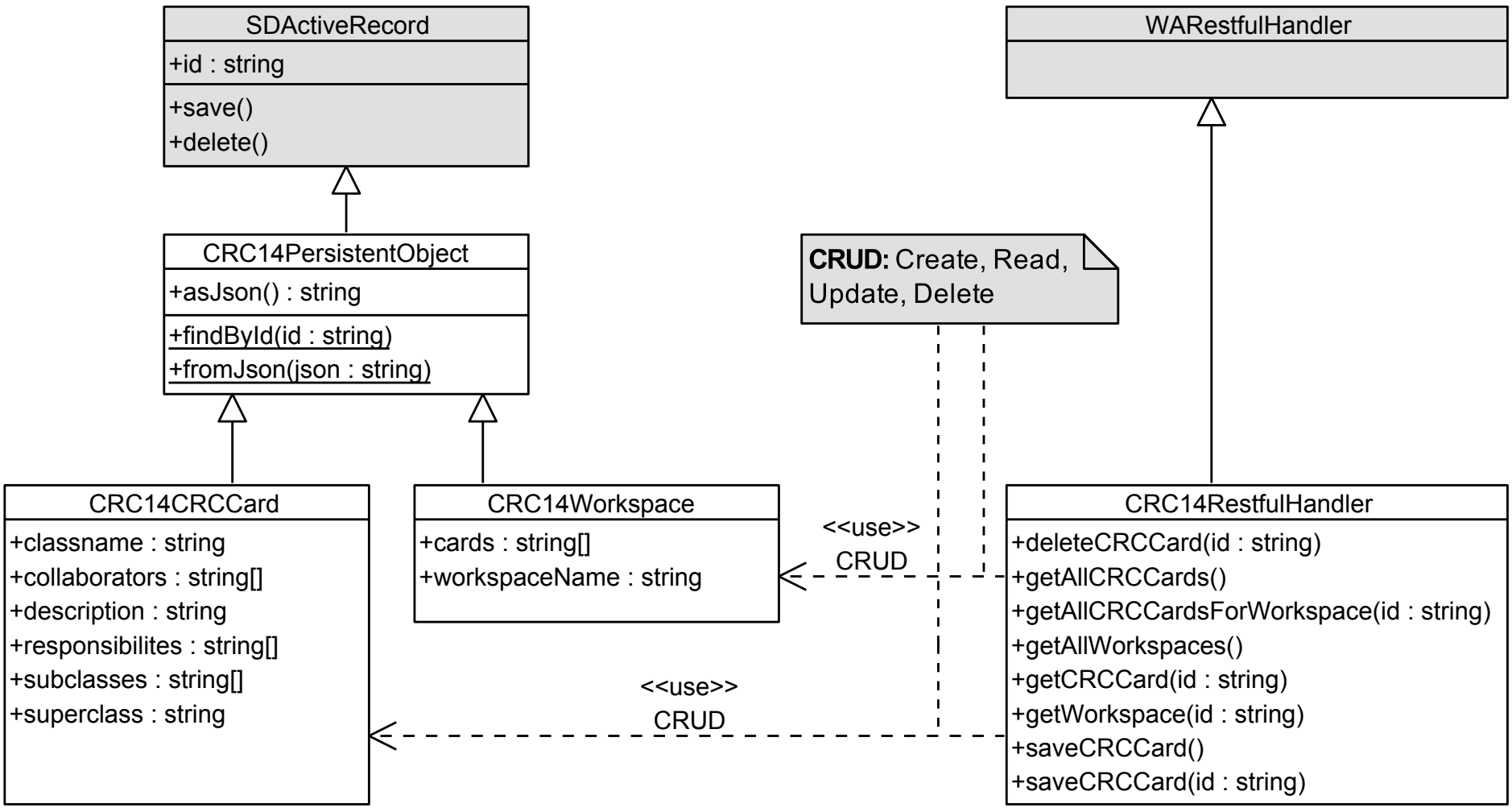
Seaside Server

- Daten persistieren
- REST-API bereitstellen
- Dateien ausliefern
- Session verwalten, Login
- Sicherheit gewährleisten

JavaScript Client

- Navigation
- Benutzerinteraktion
- Visualisierung der CRC und User Stories (Templates rendern)
- Erste Datenverifikation

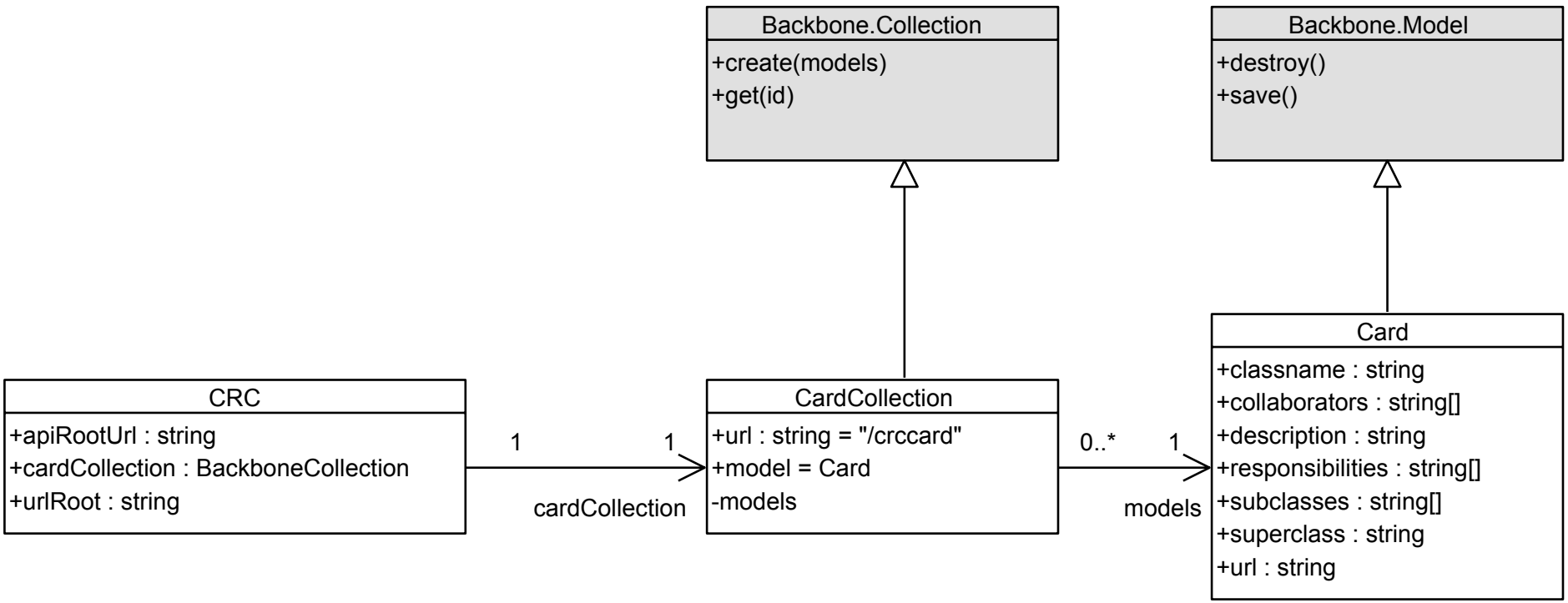
Architektur: Backend



Handout only: Architektur: Backend

- `SDActiveRecord` stellt Sandstone-Methoden bereit (z.B. Speichern, Löschen von Datensätzen)
- `CRC14PersistentObject` konvertiert von und zu JSON
- `WARestfulHandler` muss bei *Seaside REST* abgeleitet werden
- `CRC14RestfulHandler` definiert GET-, POST- und DELETE-HTTP-Handler, jeweils auf einer bestimmten URL
- Collaborators, Subclasses werden per Klassenname (String) referenziert: es können auch noch nicht-existierende Klassen referenziert werden.

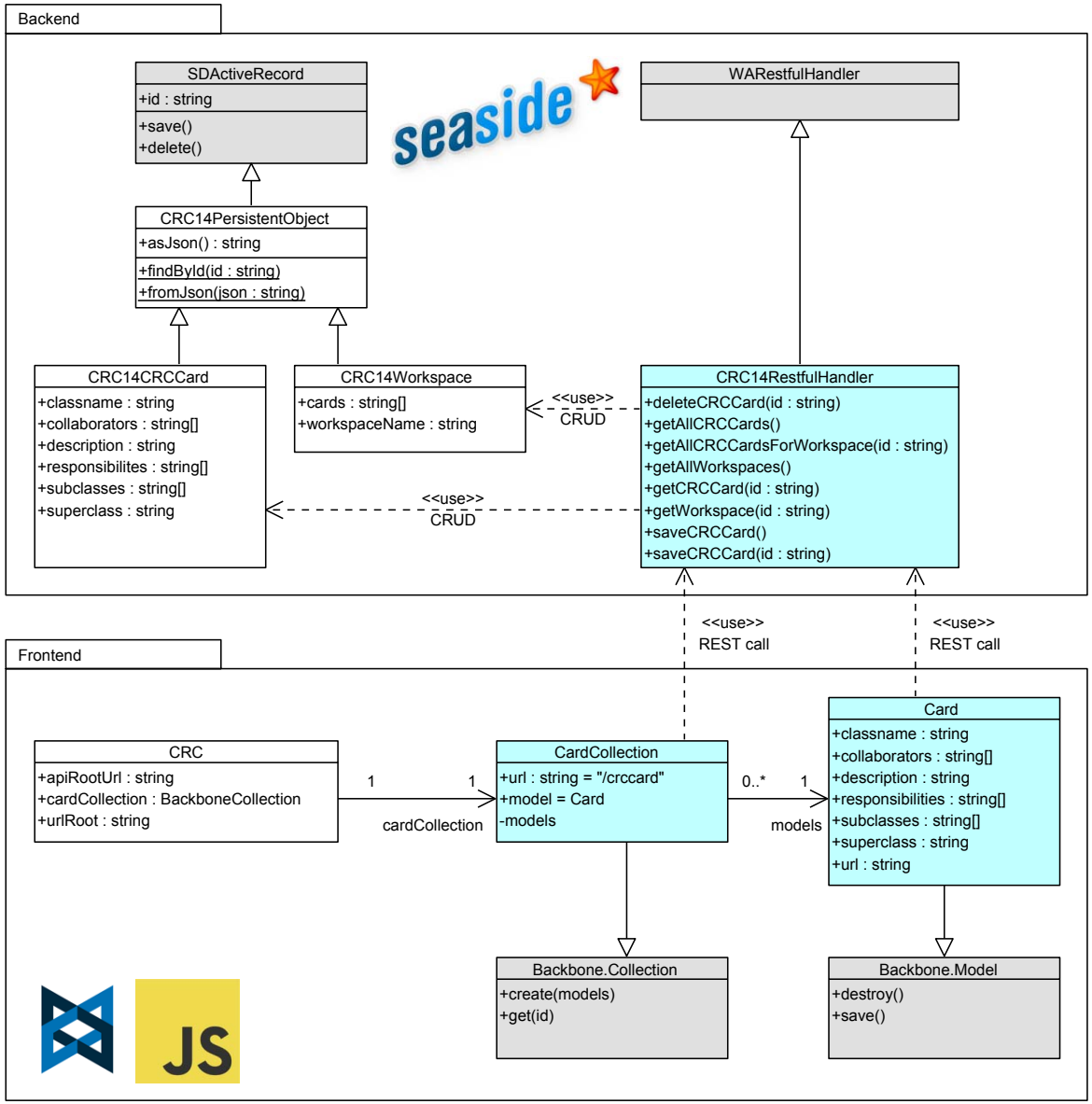
Architektur: Frontend



Handout only: Architektur: Frontend

- Identische Datenstrukturen in Frontend und Backend
- Frontend besitzt jederzeit (lokal) alle Daten
  (später: nur die Daten, die der Benutzer sehen darf)
- Backbone übernimmt Synchronisierung mit dem Backend:
  lokale Änderungen an den Daten werden automatisch an das Backend geschickt
- Hinweis: nicht alle Klassen sind hier aufgeführt

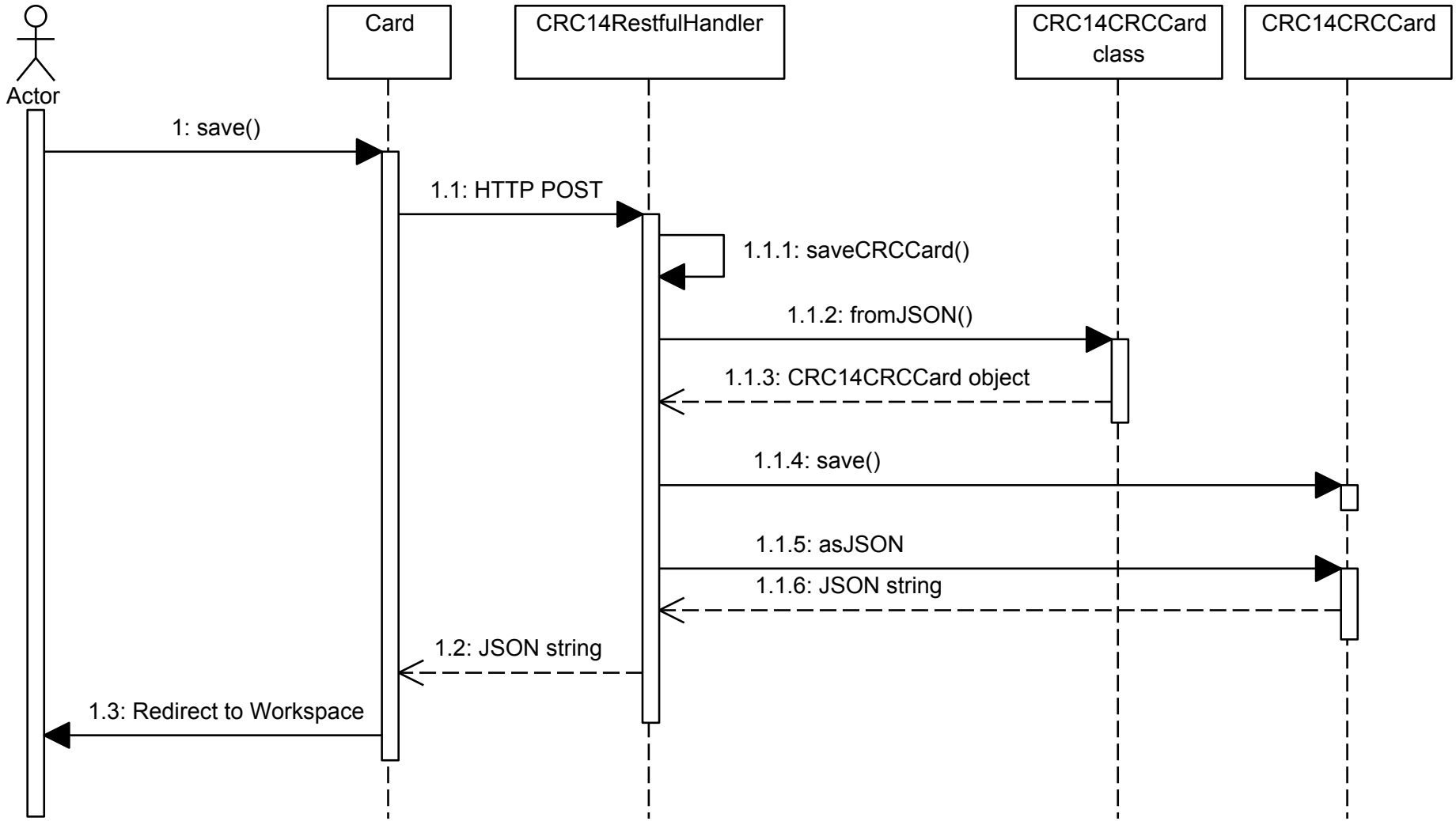
Frontend und Backend



Handout only: Frontend und Backend

- `CRC14RestfulHandler`, `CardCollection`, `Card` (blau eingefärbt) sind ein remote **Proxy** für `CRC14CRCCard`
- Kommunikation geht immer vom Frontend aus (muss regelmäßig aktualisieren)
- Kommunikation zwischen Frontend und Backend mittels REST und JSON

Interaktion: CRC-Karte speichern



Handout only: Interaktion: CRC-Karte speichern

- Alle REST-Aufrufe landen bei `CRC14RestfulHandler`
- `CRC14RestfulHandler` stellt fest, dass es sich um *Speichern einer CRC-Karte* handelte (anhand URL, HTTP-POST) und ruft `saveCRCCard` auf
- JSON-String wird zu `CRC14CRCCard` konvertiert und gespeichert
- Verifikation: Gespeicherte Karte wird wieder zu JSON konvertiert und an das Frontend geschickt (später evtl. mit Fehlercode)

Überblick

Anforderungen an den Entwicklungsprozess

- Passend für kleine Teams
- Leichtgewichtig: wenig Mehraufwand für Prozessmanagement
- Entwicklerorientiert
- Tests beinhalten
- Iterativ zur Motivation, Risikominimierung und für schnelles Feedback
- Anpassbar an unterschiedliche Stundenpläne
- Wenig Technologiewissen voraussetzen und Raum für Lernprozesse bieten

Handout only: Anforderungen an den Entwicklungsprozess

- Als 4-Mann-Team ist nur begrenzt viel Verwaltungsoverhead praktikabel
- Entwicklerorientiert, damit auch bei wenig sichtbaren Änderungen die Motivation bleibt
- Iterativ, damit es zu den wöchtentlichten Kundentreffen passt, auch bei wenig sichtbaren Änderungen die Motivation bleibt und das Risiko minimiert wird, dass über einen langen Zeitraum etwas unnötiges entwickelt wird.
- Wenig Technologiewissen voraussetzen heißt, dass Raum für unerwartete Verständnisschwierigkeiten und Lernphasen bleibt

XP versus Scrum, RUP, Wasserfall

Wasserfall

- Zu unflexibel bzgl. Ablauf und Zeit
- Erst spät Ergebnisse

V-Modell XT

- Formalisierung und hohe Qualitätsansprüche nicht erforderlich

RUP

- Risiken waren bei unserem Problem leicht überschaubar

Cont: XP versus Scrum, RUP, Wasserfall

Scrum

- Reines Scrum hat zu feste Rollen
- Viele Ideen von Scrum übernommen
  - Aufteiluung in kurze Iterationen/ Sprints
  - Product Backlog: User Stories im Redmine
  - Knowledge Creation: Wiki zur Dokumentation
  - Sprint Backlog: für eine Iteration ausgewählte User Stories

XP

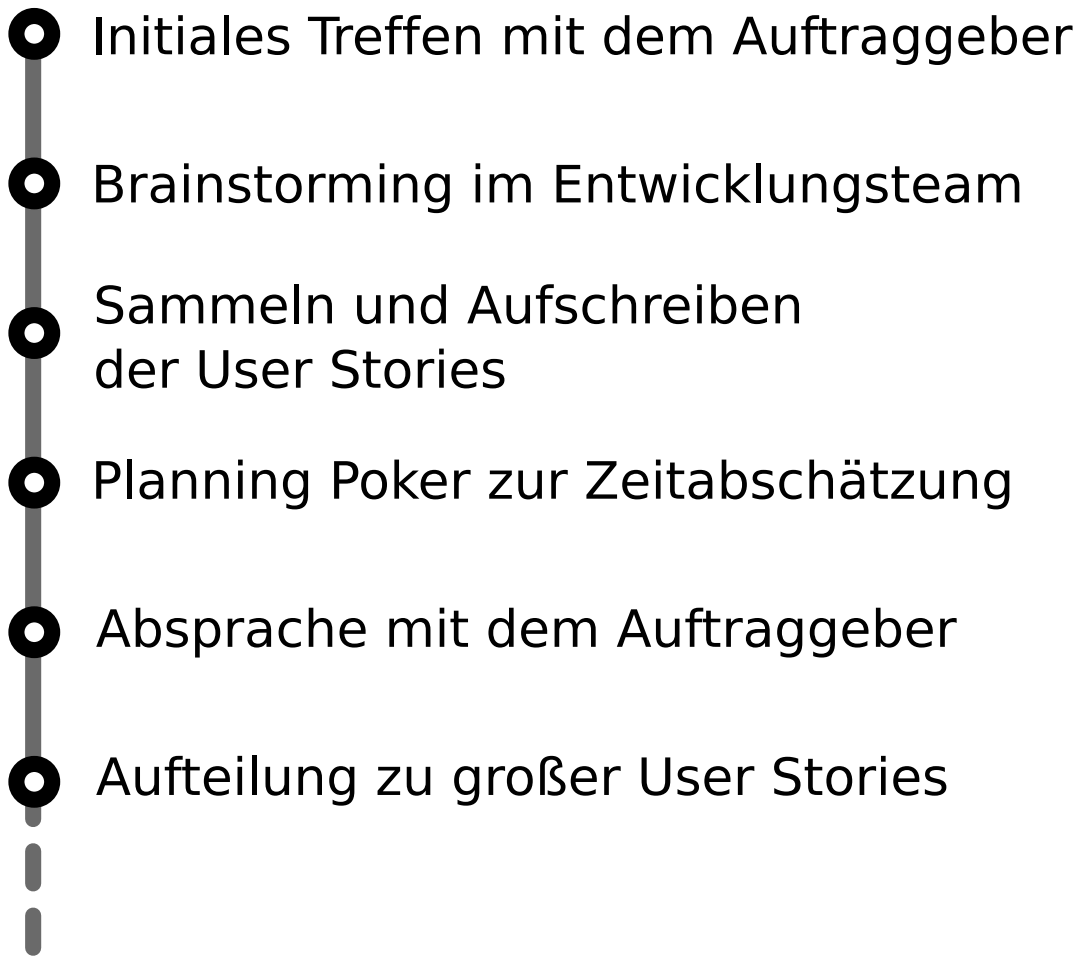
- Erfüllt mit seinen Werten und Prinzipien unsere Anforderungen
- Methoden u.a. aus Scrum helfen bei der konkreten Umsetzung

Handout only: XP versus Scrum, RUP, Wasserfall

- Reines Scrum nicht geeignet, da Rollen zu unflexibel (wie z.B. Scrum Master) wegen der unterschiedlichen Zeiten
- Auch Entwicklungsmuster wie der *Big Ball of Mud* haben uns beeinflusst ("Make it work. Make it right. Make it fast.")
- XP bietet Methodenflexibilität, sodass wir Methoden aus anderen Prozessen leicht inkorporieren konnten.

Prozess der Anforderungsanalyse

Ablauf während der ersten Woche:

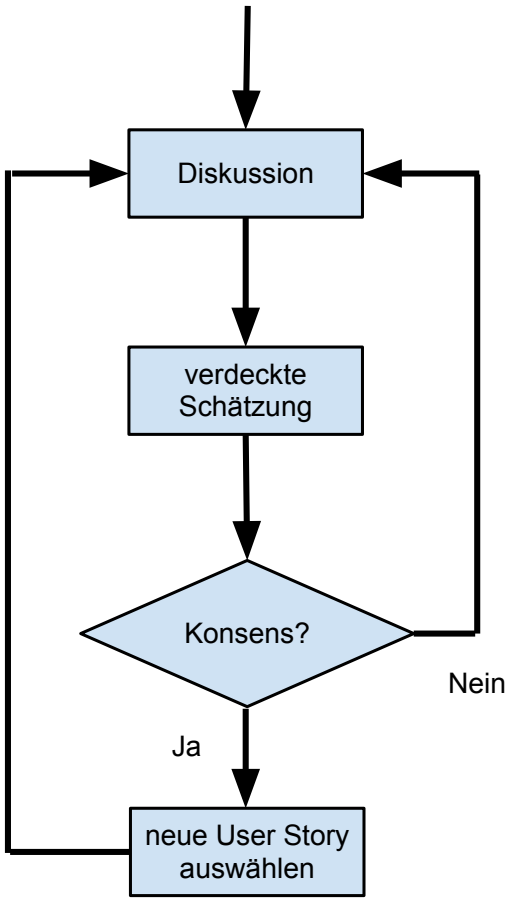


Handout only: Prozess der Anforderungsanalyse

1. Initiales Treffen mit dem Auftraggeber, um mit CRC-Karten und dem Auftrag vertraut zu werden.
2. Jedes Mitglied verfasst eine Liste mit potentiellen User Stories. Nach zehn Minuten wird abgebrochen.
3. Im Team wird über User Stories diskutiert, ob es sich um eine User Story (oder eher einen Task) handelt. Die Priorität wird festgelegt.
4. Planning Poker: siehe nächste Folien
5. Der Auftraggeber prüft Korrektheit, Vollständigkeit und Priorisierung der User Stories. Es wurden danach z.B. noch ein paar User Stories hinzugefügt.
6. Aufteilung zu großer User Stories: siehe nächste Folien

Planning Poker / Scrum Poker

- Technik zur Aufwandsschätzung von User Stories



- Klarstellung und Diskussion über User Stories
- Hilft dabei, User Stories genauer abzugrenzen
- Aufwandsschätzungen sind präziser

Quelle: `http://www.chilledpoker.com/`

Handout only: Plainning Poker / Scrum Poker

- Diskussion über Schätzungen hilft auch bei Verständnis und Abgrenzung der User Stories
- Beispiel folgt

Aufteilung zu großer User Stories

" Als **Entwickler** möchte ich bestehende CRC-Karten bearbeiten. "

- Zeitabschätzung beim Planning Poker brachten sehr unterschiedliche Ergebnisse
- User Story zu ungenau spezifiziert: was gehört alles dazu, was nicht?
- Psychologischer Aspeket: schafft das Team die User Story zeitlich nicht mehr, hat es es das Gefühl gar nichts geschafft zu haben.

User Story: CRC-Karte bearbeiten

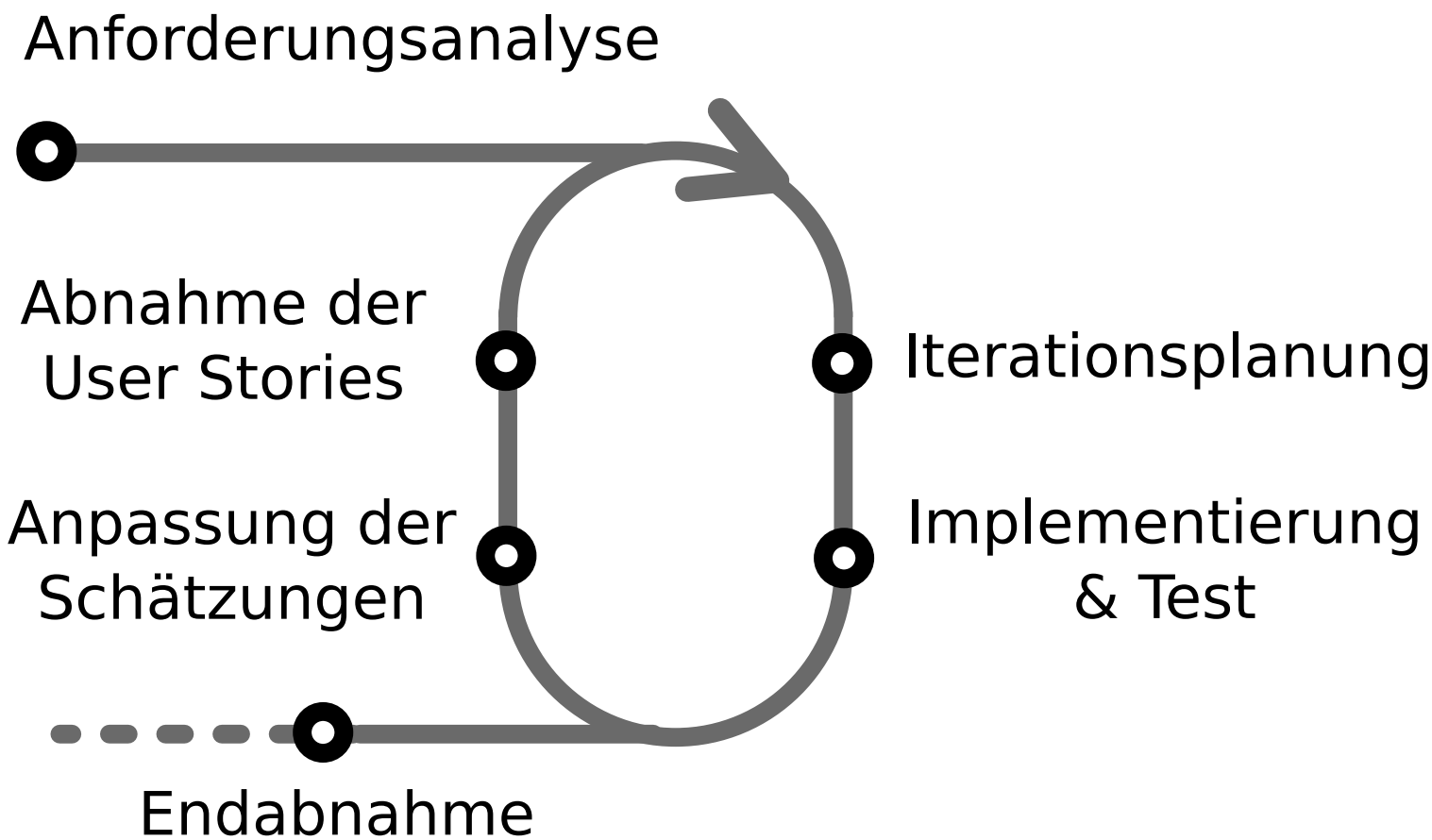
" Als **Entwickler** möchte ich CRC-Karten zu einem Workspace hinzufügen. "

" Als **Entwickler** möchte ich CRC-Karten ausfüllen (Class name, Description, Collaborators, Responsibility, Subclasses, Superclass). "

" Als **Entwickler** möchte ich CRC-Karten speichern. "

Überblick

Der Entwicklungsprozess aus zeitlicher Sicht



Handout only: Der Entwicklungsprozess aus zeitlicher Sicht

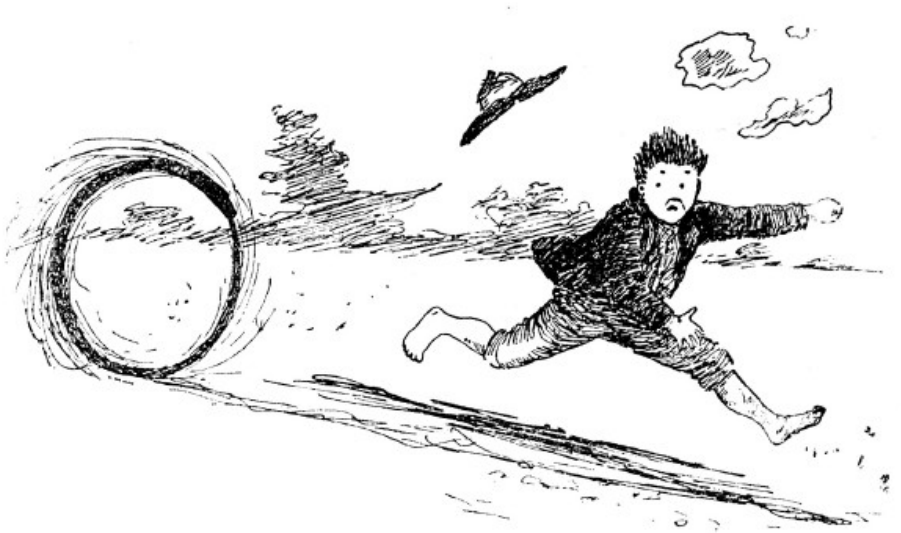
Die Punkte werden auf den nächsten Folien genauer vorgestellt.

0. Anforderungsanalyse: Projekt verstehen, Aufwand Planen - schon vorgestellt
1. Iterationsplanung: Mit dem Kunde festlegen, welche User Stories in nächsten Iteration bearbeitet werden (Planning Game)
2. Implementierung und Tests: Arbeiten, UserStories umsetzen
3. Anpassung der Schätzungen: Aufwandsschätzung und -auswertung, ggf. für weitere User Stories anpassen
4. Abnahme der User Stories: Umsetzung dem Kunden vorstellen, Rückmeldung einholen
5. Endabnahme: Projekt als Ganzes dem Kunden übergeben

1., 4. und 5. sind Kundengespräche

Kundengespräche: Iterationsplanung

- Wöchentliche Treffen mit dem Auftraggeber
- Vorstellung und Diskussion über entwickelte Features
- Auswahl von User Stories nach Priorität und geschätztem Aufwand
- Sustainable Development: meist 8 Stunden pro Woche



Quelle: `http://www.volatileprototypes.com/projects/hoopsnake/`

Handout only: Sustainable Development

- Woche 1 und Woche 2: 8 Stunden pro Woche
- Woche 3: viel andere Arbeit, sodass 8 Stunden nicht schaffbar waren
  $\rightarrow$ Fertigstellung einiger User Stories erst in der nächsten Iteration
- Danach: Arbeitszeit besser im Voraus geplant (z.B. nur wenige User Stories während der Vorbereitung des Vortrags)

Redmine

Nutzung:

- Ticket-Verwaltung
- Roadmap
- Wiki

Fazit:

- Aufwand: Dokumentation aller Handlungen
- Disziplin notwendig: sofortiges Dokumentieren von Änderungen
- Dokumentation des Entwicklungsprozess: “Projektgedächtnis”
- Übersicht: Was muss noch gemacht werden?

→ gute Entscheidung

Handout only: Redmine

Ticket-Verwaltung:

- Dokumentation von User Stories und Aufgaben
- Informationsgrundlage für Kundengespräche
- Aufwandsplanung und Aufwandsdokumentation
- Ein *Defect* erfordert bei uns in jedem Fall erst einen Test.

Roadmap:

- Definition der einzelnen Iterationen
- Iterationsplanung: Zuteilen der Userstories, Aufgaben
- Übersicht noch offener, alter Aufgaben

Wiki:

- Kurz-Spezifikation von Rest-/-Datenbankschnittstelle
- Implementierungsdetails, Seaside-Konfiguration
- Archivierung von Protokollen über Kundengespräche
- Coding Standards für Smalltalk und JavaScript

Tests

- Anfangs nur Regressiontests
  - Test-First ohne Wissen über das System (Seaside, Selenium) ist schwierig
  - Grundgerüst lässt sich schwer TDD testen
- TDD ab der zweiten Iteration
  - Keinen unnötigen Code schreiben (YAGNI)
  - Zur Anforderungsdefinition

Quelle: `http://www.metatags.org/solution`

Handout only: Tests

- Verschiedene Testtypen (werden im Folgenden vorgestellt):
  - Unittests
  - Integrationstests
    (REST API, DB API)
  - Akzeptanztests beim Kunden

Unittest: CRC-Karte im Backend speichern

Set Up:	CRC-Karte erstellen
Execute:	CRC-Karte speichern (Sandstone DB)
Verify:	CRC-Karte aus der Datenbank holen und Inhalt prüfen
Tear Down:	CRC-Karte aus der Datenbank löschen

Integrationstest: CRC-Karte im Frontend speichern

Set Up: Eingabeformular für neue CRC-Karte per Selenium öffnen

Execute:
1. Forumularfelder ausfüllen
2. *Speichern* Button klicken

Verify: CRC-Karte aus der Datenbank holen und Inhalt prüfen

Tear Down: CRC-Karte aus der Datenbank löschen

Akzeptanztest: Eingabemaske für CRC-Karten

- Vorführung einer Funktionalität (Karte erstellen, ausfüllen, speichern) beim Auftraggeber
- Kundenwunsch: andere Anordnung der Formularfelder

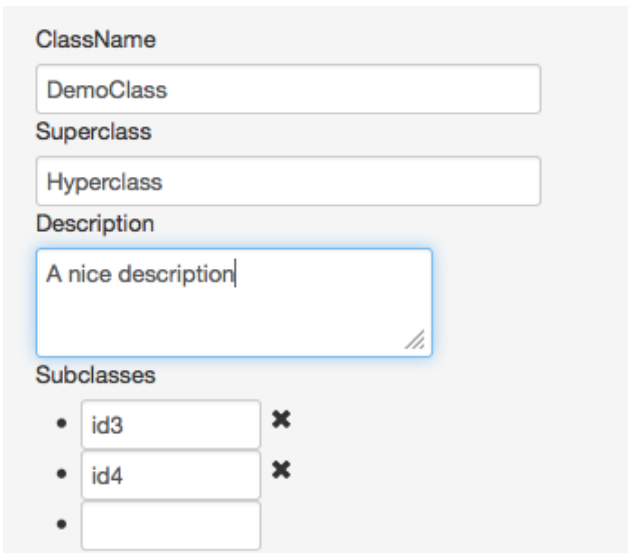


Abbildung: Vorher

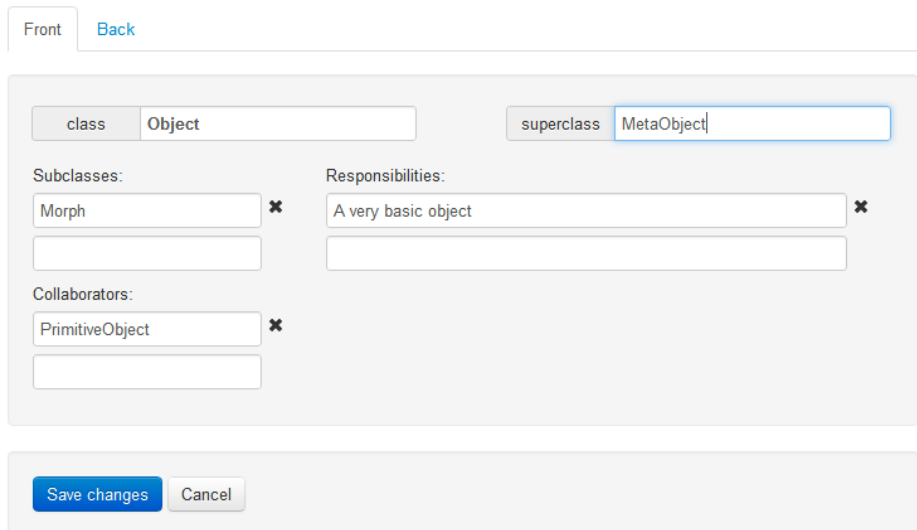


Abbildung: Nachher

Tests

Problem

JavaScript-Unittests sind nicht mit Seaside oder Selenium möglich

Lösung

Separates Testingframework (QUnit)

- Neue Seite für JavaScript Unittests
- Ein Seleniumtest um alle Tests in der Testsuite zu prüfen

n-Version Programming ($n = 2$)

Experiment: Zwei Gruppen (Paare) implementieren die gleiche Funktion.

Ergebnis der n-Version-Programmierung

- Sehr ähnliche Implementierungen (Datenstrukturen, Referenzen)
- Unterschiedliche Datenbanken (Sandstone DB, JSON Datenbank)

Fazit

- Entwicklungszeit verdoppelt sich
- Disziplin notwendig: ein Backend wird später größtenteils verworfen
- Zufriedenheit mit dem Backend im ganzen Team

Handout only: n-Version Programming ($n = 2$)

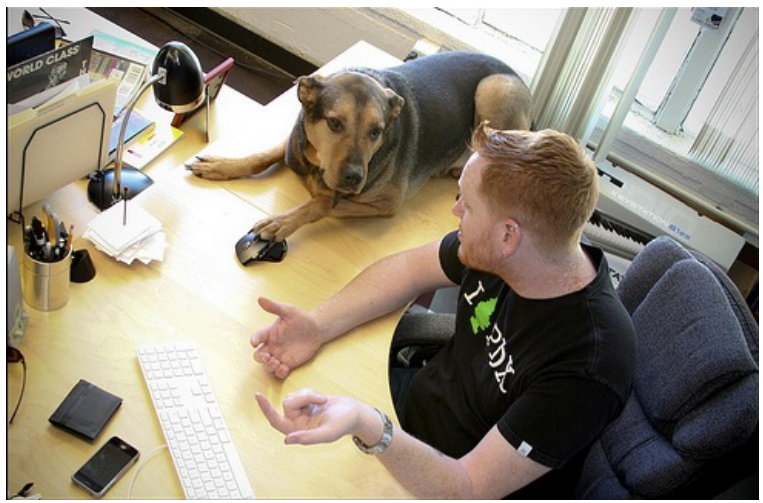
- Parallele, unabhängige Entwicklung zweier Entwicklungsteams
- Idee: Qualitativ hochwertiges Backend, Fehler und spätere größere Änderungen vermeiden
- Pair Programming zeiteffizienter
- Nutzung nur bei speziellen Fällen sinnvoll (Hohe Sicherheit, Neue Technologien, Alternativen finden ..)

Beispiele für *sehr ähnliche Implementierungen*

- Referenzierung von Collaborators, Subclasses über Namen der Klassen (ermöglichst Referenzierung nicht-existierender Klassen)
- Speicherung von Referenzen auf CRC-Karten im Workspace (Strings: Sandstone DB IDs)

Pair Programming

- Breitere Kenntnisse des Systems im Team
  → Verhindert Expertenwissen, fördert Collective Code Ownership
- Kein Pair Programming bei der Entwicklung des Frontend-Grundgerüsts
  → Nur eine Person kennt sich aus.
- Lernen neuer Techniken, Programmiersprachen und Frameworks
- Zeitersparnis, vor allem bei Selenium-Tests



Quelle: `http://swik.net/Rails+Projects`

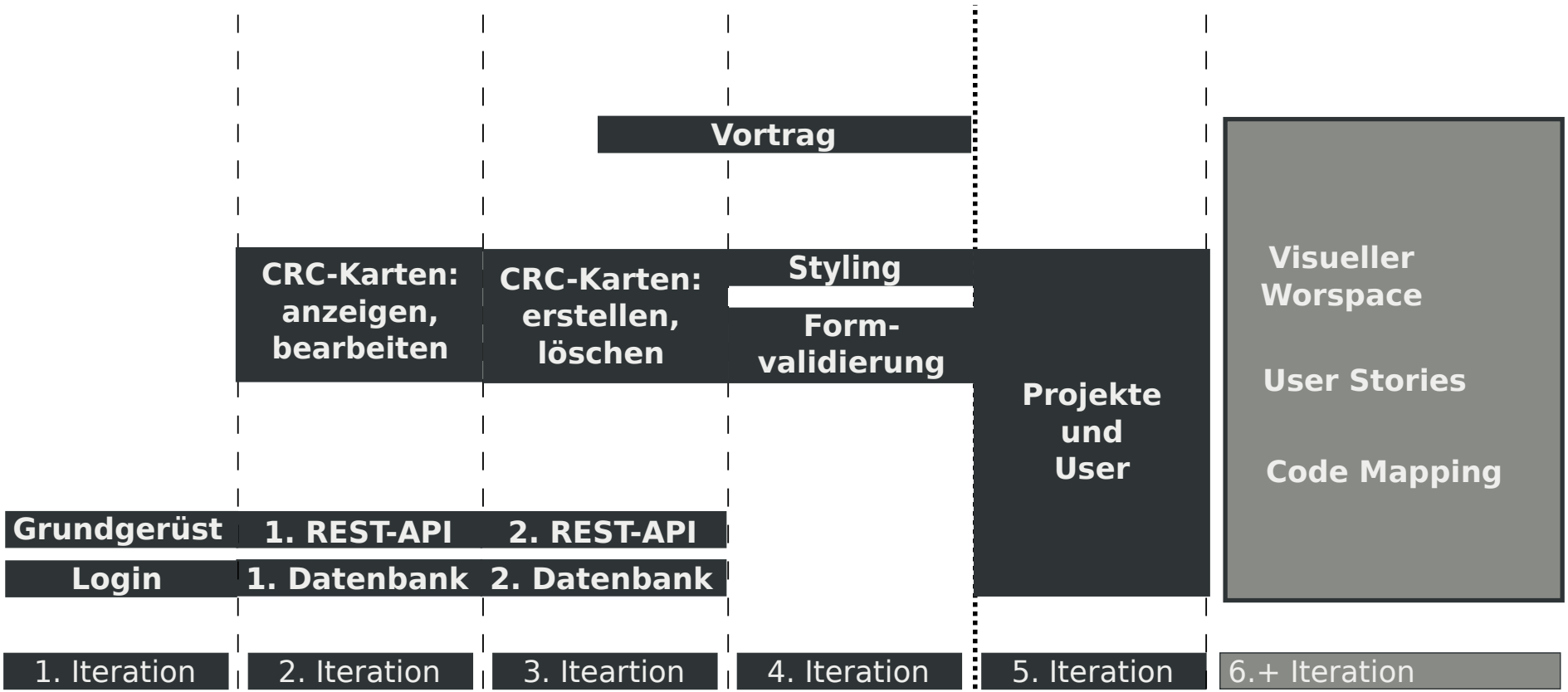
Handout only: Pair Programming

Folgende Komponenten wurden mit Pair Programming entwickelt

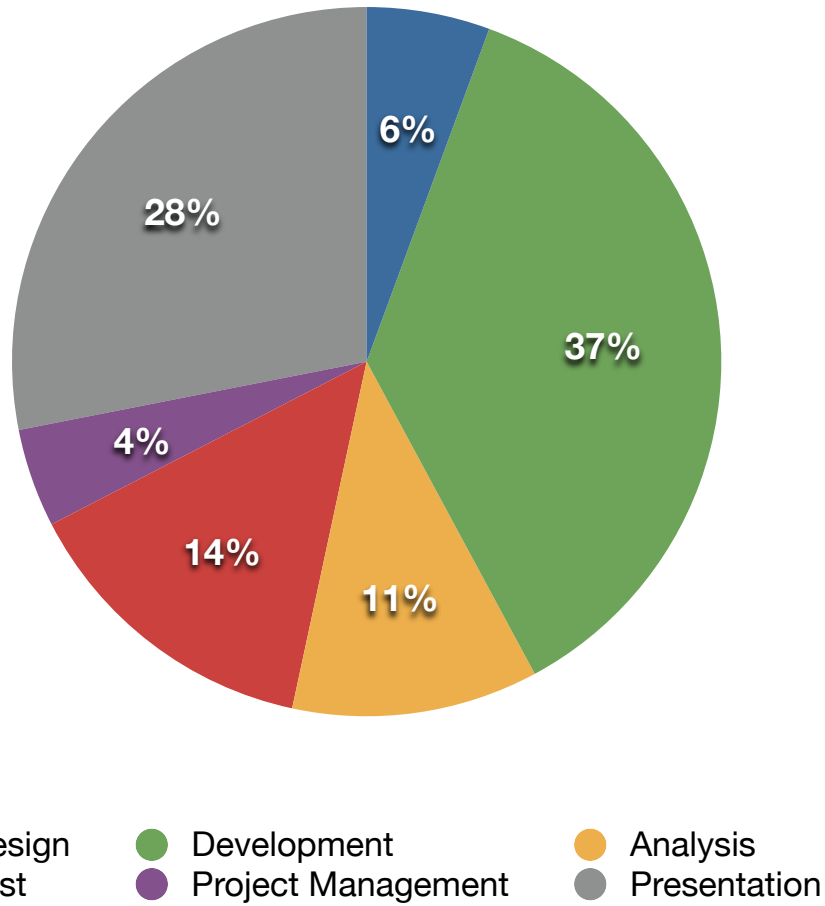
- Teile des Backends (z.B. REST-Handler)
- CRC-Eingabeformulars und Workspace-Liste im Frontend
- Web-Tests (v.a. Selenium-Tests, z.B. OpenID-Login-Test)
- Stellenweise wurde bewusst auf Pair Programming verzichtet, z.B. beim Stylen und anderen einfachen Aufgaben.

Überblick

Ablauf und Ausblick



Aufwandsstatistik



Vergleich mit SWA

- Gleiches Team
- In SWA keine Tests, sehr viel Zeit mit Spielen verbracht
- Keiner hatte Erfahrungen mit dem genutzten System
- In SWT
  - Strukturierter, bewusster Entwicklungsprozess
  - Weniger visuelles Projekt, bei dem viel Arbeit oft nicht sichtbar ist (OpenID, Datenmodell . . .)
  - Absprachen mit den Kunden - weniger eigene Entscheidung, was getan werden soll
  - Erfahrung mit JavaScript/CSS beschleunigt Entwicklung
  - Kleinschrittigeres Vorgehen und wenig Planung für die Zukunft (*YAGNI*)

Quellen

- Projektmanager: Danke an Andreas Rau
- Developer: Matthias Springer