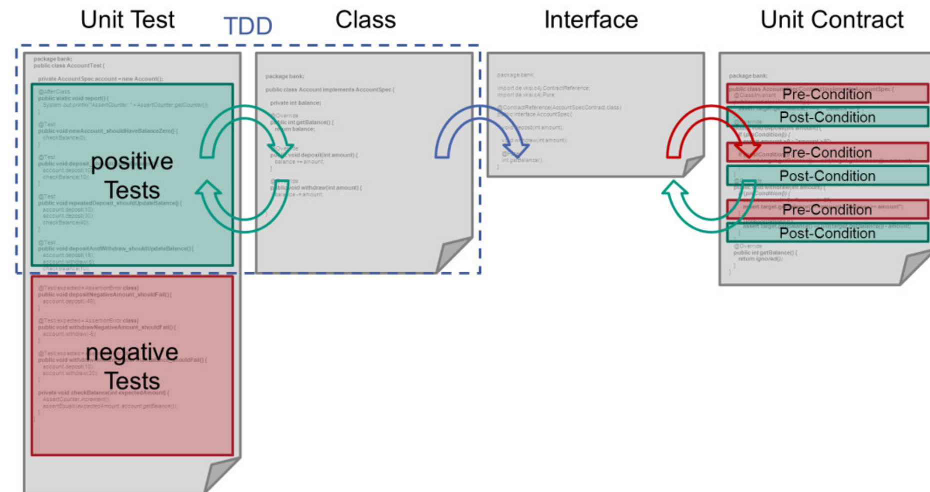
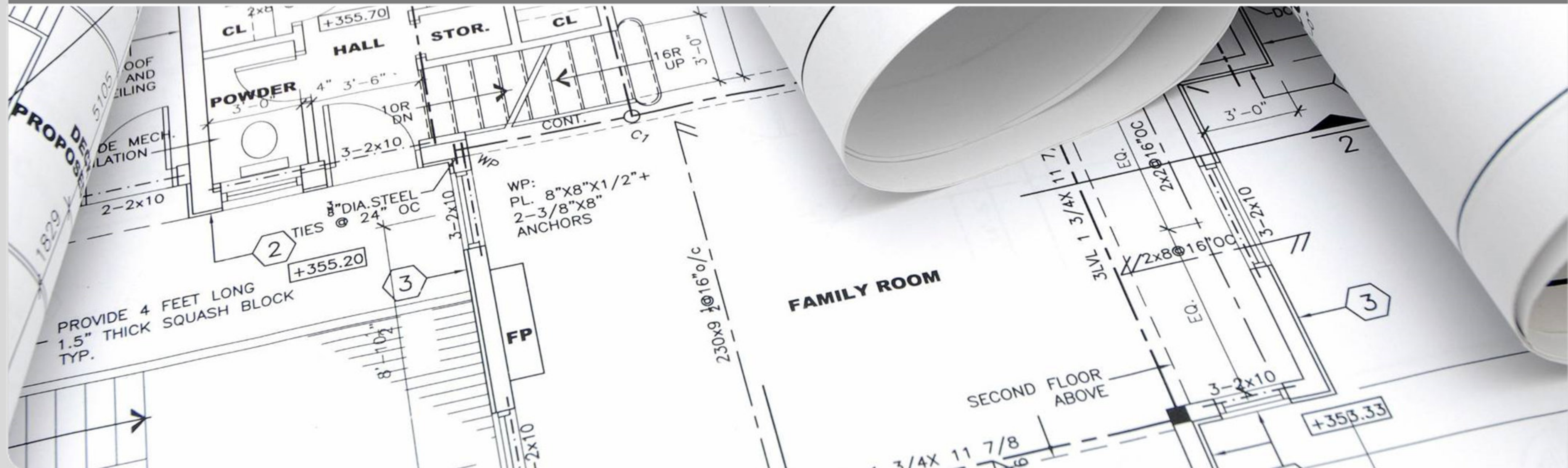
VKSI Verein der Karlsruher Software-Ingenieure



TDD with Contracts

Karlsruher Entwicklertag 2014
Hagen Buchwald, andrena objects
Prof. R. Küstermann, DHBW Karlsruhe

VKSI SIG C4J - Special Interest Group „Contracts for Java“ des Vereins Karlsruher Software Ingenieure



Inhaltsübersicht
TDD with Contracts

1. Prolog

2. Was ist TDD with Contracts?

3. Wie funktioniert TDD with Contracts (Demo)?

4. Welche Erfahrungen gibt es mit TDD with Contracts?

5. Epilog

6. Fragen & Antworten

Was ist Objektorientierte Programmierung?

Definition nach Alan Kay (2003):

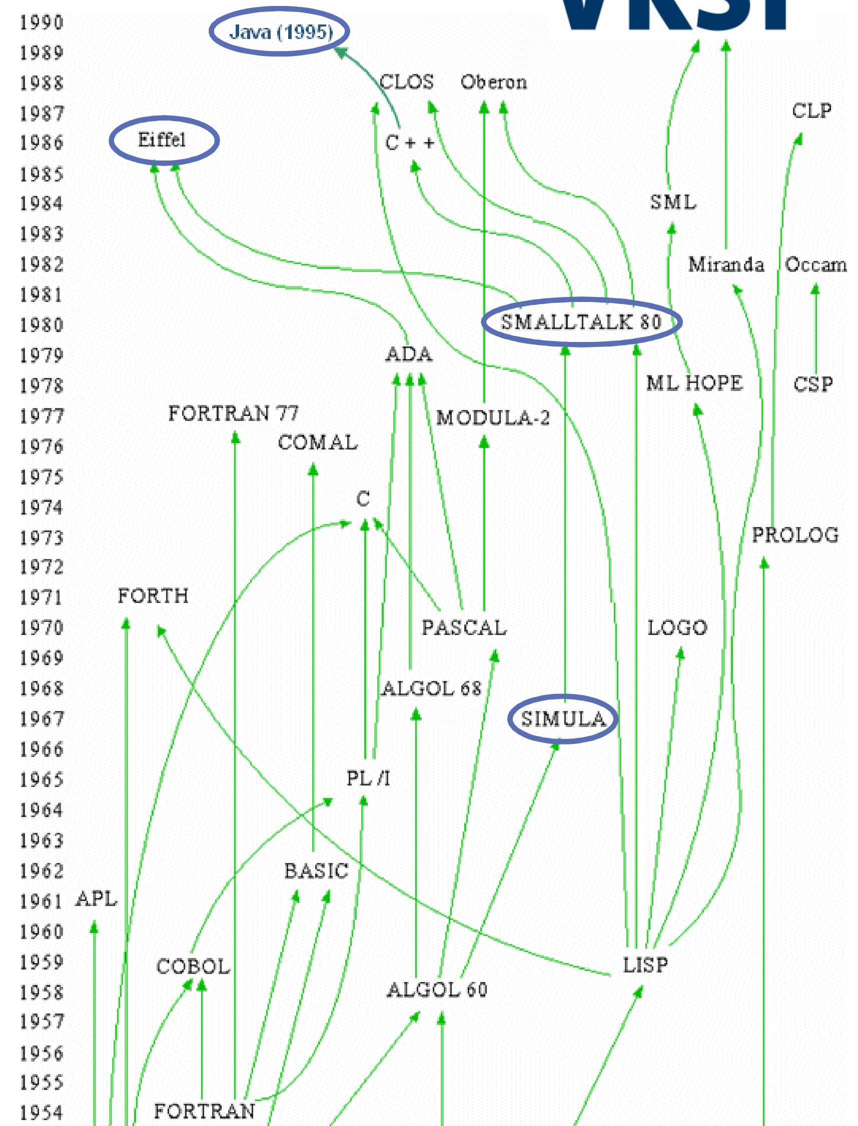
“OOP to me means only messaging, local retention and protection and hiding of state-process, and extreme late-binding of all things. It can be done in Smalltalk and in LISP. There are possibly other systems in which this is possible, but I’m not aware of them. ”

“One of the things I should have mentioned is that there were two main paths that were catalysed by Simula.

The early one was the bio/net non-data-procedure route that I took.

The other one, which came a little later as an objects of study was abstract data types, and this got much more play.”

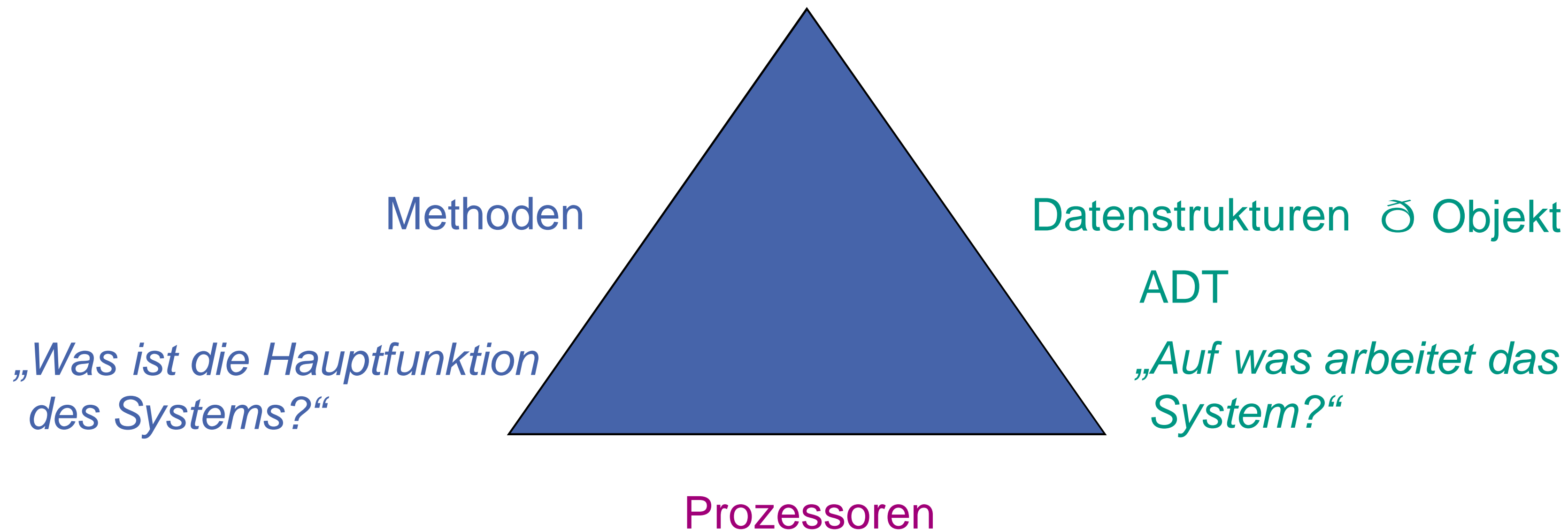
Quelle: http://www.purl.org/stefan_ram/pub/doc_kay_oop_en

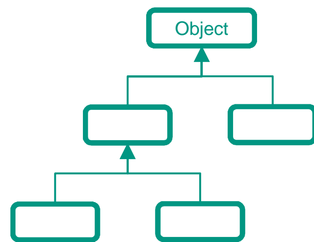


Objekt-Orientierte Programmierung (OOP).
Definition nach Bertrand Meyer (1988)

VKSI

Ein Softwaresystem besteht aus Prozessoren,
die Methoden auf Datenstrukturen ausführen.





OOP ist die Erstellung von Softwaresystemen
als strukturierte Sammlung von Implementierungen
Abstrakter Datentypen (ADTs).

Quelle: Bertrand Meyer, Object-Oriented Software Construction, 1988

Was ist ein Abstrakter Datentyp (ADT)?

Ein Abstrakter Datentyp (Abstract Data Type) besteht aus vier Teilen:

1) TYP	`1) AccountSpec`

2) METHODEN	`2) void deposit(int amount)`
	`void withdraw(int amount)`
	`int getBalance()`

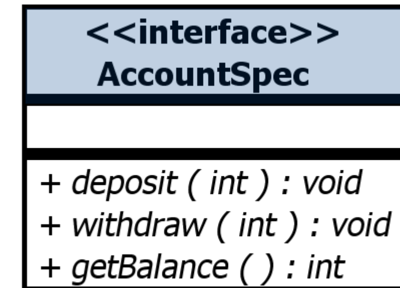
3) VORBEDINGUNGEN	`3) deposit: amount > 0`
	`withdraw: amount > 0`
	`getBalance >= amount`
	`getBalance: ---`

4) NACHBEDINGUNGEN	`4) deposit: getBalance == old getBalance + amount`
	`withdraw: getBalance == old getBalance - amount`
	`getBalance: result >= 0`

In Anlehnung an: Bertrand Meyer, Object-Oriented Software Construction, 1988

Kann man in Java Objekte als Implementierungen Abstrakter Datentypen beschreiben?

```
TYP
    AccountSpec

METHODEN
    void deposit(int amount)
    void withdraw(int amount)
    int getBalance()

VORBEDINGUNGEN
    deposit:      amount > 0
    withdraw:     amount > 0
                  getBalance >= amount
    getBalance:   ---

NACHBEDINGUNGEN
    deposit:      getBalance == old getBalance + amount
    withdraw:     getBalance == old getBalance – amount
    getBalance:   result >= 0
```



Welche Rolle spielen Interfaces in Java?

- Joshua Bloch spricht in seinem Buch „Effective Java“ (Second Edition, 2008) insgesamt 78 Punkte (Items) an, wie man mit Java programmieren sollte, um hohe Qualität zu erzeugen.

  - Item 19: Use interfaces only to define types

    „When a class implements an interface, the interface serves as a type that can be used to instances of the class. It is inappropriate to define an interface for any other purpose.“

  - Item 40: Design method signatures carefully

    „Use interfaces rather than classes as parameter types.“

  - Item 52: Refer to objects by their interfaces

    „If appropriate interfaces types exist, then parameters, return values, variables and fields should all be declared using interfaces types. If you get into the habit of using interfaces as types, your program will be much more flexible.“

Inhaltsübersicht
TDD with Contracts

1. Prolog
2. Was ist TDD with Contracts?
3. Wie funktioniert TDD with Contracts (Demo)?
4. Welche Erfahrungen gibt es mit TDD with Contracts?
5. Epilog
6. Fragen & Antworten

Prinzip eines Vertrags: Rechte und Pflichten. Die Rechte des Kunden sind die Pflichten des Anbieters – und umgekehrt!

Kunde	Anbieter
Rechte	Rechte
Pflichten	Pflichten

Beispiel für einen Vertrag:
Ein Kunde kauft ein Brötchen in einer Bäkerei.

VKSI

Kunde		Anbieter
Rechte „Ich erhalte ein Brötchen.“		Rechte „Ich erhalte 50 Cent.“
Pflichten „Ich muss 50 Cent zahlen.“		Pflichten „Ich muss ein Brötchen liefern.“

Es reicht aus, den Vertrag auf einer der beiden beteiligten Seiten zu definieren und seine Einhaltung zu überprüfen.

VKSI

Kunde

Rechte

Pflichten

Anbieter

Rechte

Pflichten

In unserem Beispiel: Es reicht aus, den Vertrag auf Seiten des Bäckers zu definieren und seine Einhaltung zu überprüfen.

VKSI

Kunde	Anbieter
Rechte „Ich erhalte ein Brötchen."	**Rechte** „Ich erhalte 50 Cent."
Pflichten „Ich muss 50 Cent zahlen."	**Pflichten** „Ich muss ein Brötchen liefern."

© H. Buchwald, Prof. R. Küstermann

ADT `AccountSpec` : Die Vor- und Nachbedingungen stellen einen Vertrag dar. Es genügt, diesen Vertrag auf Anbieterseite zu definieren und zu prüfen.

Kunde | Anbieter

Kunde: Rechte, Pflichten

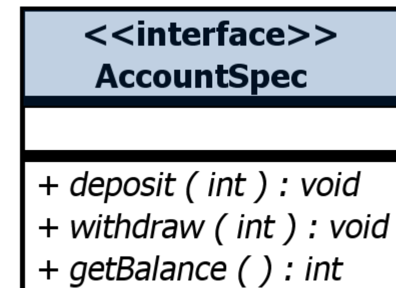
```
1) AccountSpec

2) void deposit(int amount)
   void withdraw(int amount)
   int getBalance()

3) deposit:       amount > 0
   withdraw:      amount > 0
                  getBalance >= amount
   getBalance:    ---

4) deposit:       getBalance ==
                  old getBalance + amount
   withdraw:      getBalance ==
                  old getBalance - amount
   getBalance:    result >= 0
```

Anbieter: Rechte, Pflichten

Die *Vorbedingungen* definieren die *Rechte des Anbieters* (und damit die *Pflichten des Kunden*).
Die *Nachbedingungen* definieren die *Pflichten des Anbieters* (und damit die *Rechte des Kunden*).

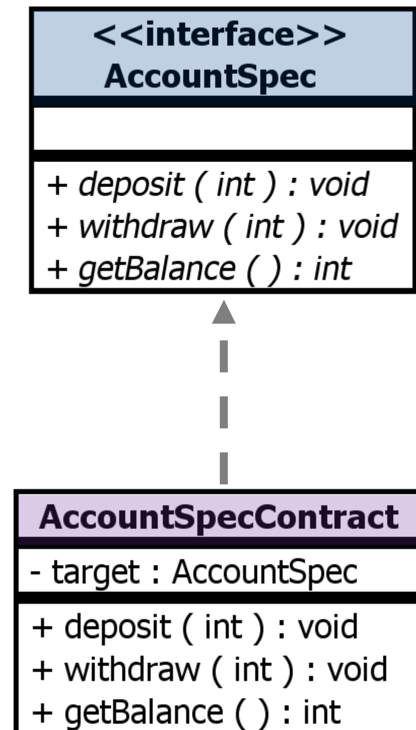
Beispiel für einen Softwarevertrag
Der Typ `AccountSpec`

```
TYPE
    AccountSpec

METHODS
    void deposit(int amount)
    void withdraw(int amount)
    int getBalance()

PRE-CONDITIONS
    deposit:      amount > 0
    withdraw:     amount > 0
                  getBalance >= amount
    getBalance:   ---

POST-CONDITIONS
    deposit:      getBalance == old getBalance + amount
    withdraw:     getBalance == old getBalance - amount
    getBalance:   result >= 0
```



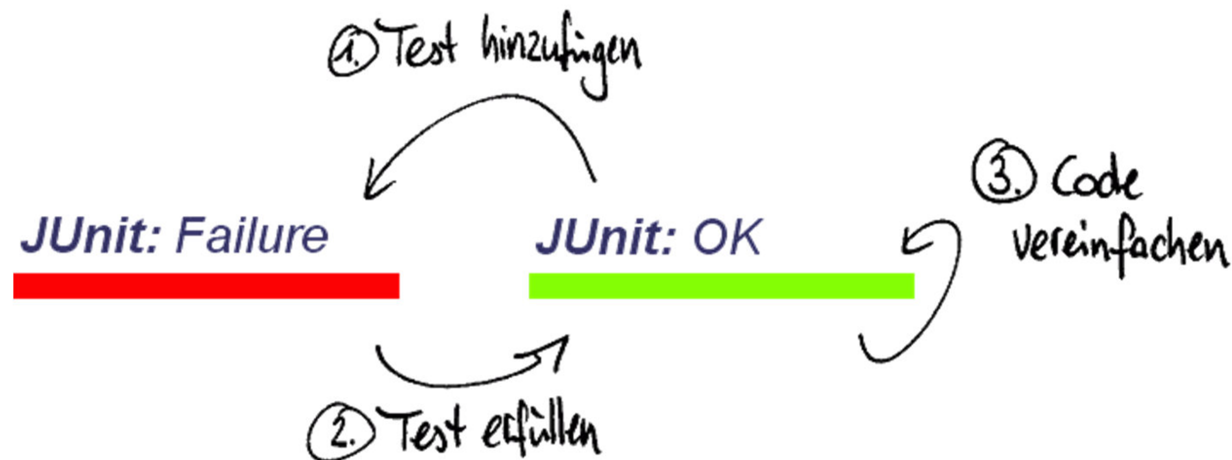
Der Typ `AccountSpec` formuliert als Abstrakter Datentyp (ADT) in Java.

```
TYPE
    AccountSpec

METHODS
    void deposit(int amount)
    void withdraw(int amount)
    int getBalance()

PRE-CONDITIONS
    deposit:      amount > 0
    withdraw:     amount > 0
                  getBalance >= amount
    getBalance:   ---

POST-CONDITIONS
    deposit:      getBalance == old getBalance + amount
    withdraw:     getBalance == old getBalance – amount
    getBalance:   result >= 0
```



TDD Cycle

VKSI



make it red: schreibe einen Test, der fehlschlägt

make it green: schreibe gerade so viel Code, dass alle Tests laufen

make it right: beseitige duplizierten Code und andere *code smells*

TDD with Contracts

VKSI

Unit Test | TDD | Class | Interface | Unit Contract

positive Tests

negative Tests

Pre-Condition

Post-Condition

Pre-Condition

Post-Condition

Pre-Condition

Post-Condition

Clean Code

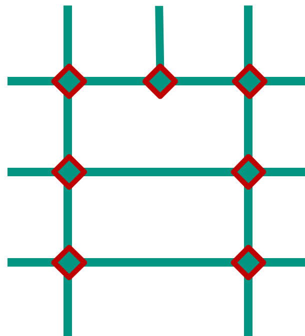
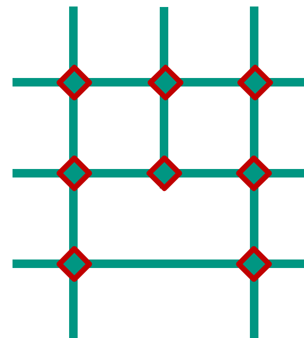
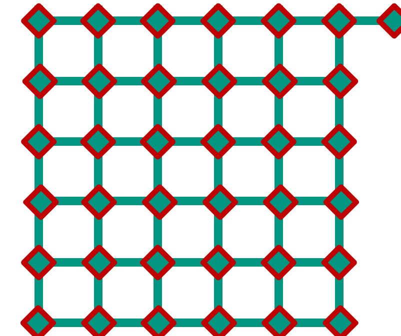
Clean Contract

Klassen-Ebene (Implementierung)

Typen-Ebene (Abstrakter Datentyp)

Vergleich von TDD, DbC und TDD with Contracts am Beispiel der Klasse `Account`

VKSI

TDD	DbC	TDD with Contracts
		
Sicherheitsnetz bestehend aus 7 assert-Statements	Sicherheitsnetz bestehend aus 8 assert-Statements	Sicherheitsnetz bestehend aus 37 assert-Statements

- **Unit Tests und Unit Contracts als äußerst feinmaschiges Sicherheitsnetz**
  - **Fehlerhafte Zustände werden bereits beim automatisierten Testen entdeckt**

Brauchen wir bessere Sicherheitsnetze?
Und falls ja: Wie hoch sind die Zusatzkosten?

Kunden-
zufriedenheit

sehr
zufrieden

Leistungs-
merkmale

Begeisterungs-
merkmale

realisierte
Qualitätseigenschaften

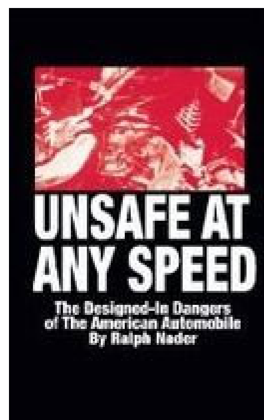
wenig

indifferent

viel

Basis-
merkmale

völlig
unzufrieden



Inhaltsübersicht
TDD with Contracts

1. Prolog
2. Was ist TDD with Contracts?
3. Wie funktioniert TDD with Contracts (Demo)?
4. Welche Erfahrungen gibt es mit TDD with Contracts?
5. Epilog
6. Fragen & Antworten

TDD with Contracts (Demo)
Der Auftrag

- Die Motivation: Die MC²-Bank möchte die Kundengruppe der Akademiker ausbauen.
- Die Strategie: Das kostenlose Studentenkonto.
- Der Auftrag: Es muss sichergestellt sein, dass die Studenten ihr Konto auf keinen Fall überziehen können.

- Die Lösung: Die Klasse **`Account`**

Bild-Quelle: https://www.sparen-online.at/fileadmin/main/pictures/header/studentenkonto.jpg

TDD with Contracts
Schritt für Schritt Demo

Unit Test | TDD | Class | Interface | Unit Contract

positive Tests

negative Tests

Pre-Condition

Post-Condition

Pre-Condition

Post-Condition

Pre-Condition

Post-Condition

Clean Code

Clean Contract

Klassen-Ebene (Implementierung)

Typen-Ebene (Abstrakter Datentyp)

Inhaltsübersicht
TDD with Contracts

1. Prolog
2. Was ist TDD with Contracts?
3. Wie funktioniert TDD with Contracts (Demo)?
4. Welche Erfahrungen gibt es mit TDD with Contracts?
5. Epilog
6. Fragen & Antworten

Contracts for Java – die erste Generation

VKSI

- C4J 1.x
  - 2006 Stockholm — bwin Poker Engine, Jonas Bergström entwickelt C4J
  - 2008 Karlsruhe — KIT Vorlesung „Programmieren I“ mit C4J (AIFB, Hagen Buchwald)
  - 2011 Karlsruhe — Diplomarbeit zum Thema Testautomatisierung mit Softwareverträgen (Yana Stoeva)
  - 2011 Karlsruhe — Entwicklertag 2011
Jonas Bergström stellt C4J vor
Keynote von Hagen Buchwald zum Thema „Design by Contract“ in Java
Podiumsdiskussion zum Thema Contracts u.a. mit Johannes Link

Testing by Contract (TbC) : PokerBots simulieren Spieler und testen die Poker-Engine bei aktivierten Verträgen

Wirkung des Einsatzes von Testing by Contract mit C4J und PokerBots auf die Qualität der bwin PokerEngine

bwin PokerEngine
ohne C4J

Fehlermeldungen
pro Betriebsmonat

bwin PokerEngine
mit C4J

Fehlermeldungen nach
38 Betriebsmonaten

Einsatz von Softwareverträgen seit 2006 bei bwin in Stockholm
Lessons Learned

VKSI

1. Pareto-Prinzip: 20% der Klassen stellen 80% der genutzten Funktionalität zur Verfügung. Es reicht aus, diese Kernklassen mit Softwareverträgen zu schützen.

2. Falsche Annahmen als Hauptfehlerquelle: 80% der bei bwin gefundenen Fehler waren auf Verletzungen von Vorbedingungen (und Klasseninvarianten) zurückzuführen. Der Grund waren falsche bzw. fehlende Annahmen darüber, wie eine Klasse genutzt werden sollte.

3. Autorenschaft schützt nicht vor falschen Annahmen: Falsche Annahmen unterliefen auch den Autoren der Klassen. Nach nur wenigen Wochen Arbeit an anderen Modulen hatten sie die Annahmen vergessen, unter denen sie die Klassen erstellt hatten.

4. Testing by Contract: Die Kombination von Softwareverträgen mit PokerBots – virtuellen Spielern, die über Nacht die aktuelle Version der PokerEngine nutzten – erwies sich als äußerst wirkungsvolles Instrument, um subtile Fehler aufzuspüren (Boundary Conditions, Edge Conditions, Corner Conditions).

5. Macht der Gewohnheit: Trotz dieser Erfolge blieb die Nutzung von Softwareverträgen auf das PokerEngine-Team beschränkt. Andere Teams zogen nicht nach.

Quelle: Interview mit Jonas Bergström, Sept. 2012, Stockholm

Contracts for Java – die zweite, agile Generation

VKSI

- C4J 6.x
  - 2012 Karlsruhe — VKSI SIG C4J Ben Romberg et al. entwickeln die agile Generation von C4J
  - 2012 Karlsruhe — Entwicklertag 2012 Vorstellung von C4J 6.0
  - 2012 Karlsruhe — DHBW SS 2012 Vorlesung „Contracts for Java“
  - 2012 München — JBFOne 2012 Vortrag „Contracts for Java“
  - 2013 München — OOP 2013 Jonas Bergström stellt TbC vor
  - 2013 Mannheim — In seiner (preisgekrönten) Masterarbeit an der HS Mannheim entwickelt Florian Meyerer ein C4J-Plugin für Eclipse (ext. Betreuer: H. Buchwald)
  - 2013 Karlsruhe — Einsatz des neuen C4J-Plugins an DHBW im SS 2013 in der Vorlesung „Contracts for Java“
  - 2013 Rapperswil — Hochschule für Technik Rapperswil (Schweiz) Studienarbeit zur Nutzung von C4J in der Lehre
  - 2013 Karlsruhe — Auf den XP Days Germany stellen Dr. Lars Alvincz und Hagen Buchwald den Ansatz „TDD with Contracts“ vor
  - 2014 Karlsruhe — DHBW Karlsruhe 2013/2014, Prof. R. Küstermann; Abschlussprojekt im 3. Studienjahr mit bewusstem Einsatz von TDD with Contracts

DHBW Karlsruhe 2014/15

DHBW
Duale Hochschule
Baden-Württemberg
Karlsruhe

Prof. R. Küstermann

- Die Duale Hochschule Baden-Württemberg (DHBW) ist mit rund 34.000 Studierenden an 12 Standorten sowie 9.000 kooperierenden Unternehmen und sozialen Einrichtungen die größte Hochschule des Landes.
- Die Studienakademie Karlsruhe, mit mehr als 3.000 Studierenden, leistet in der Technologieregion Karlsruhe einen großen Beitrag zur Sicherung des IT-Führungsnachwuchses.
- *„Im Studiengang Wirtschaftsinformatik ist TDD with Contracts ein gelungenes Beispiel für eine enge Verzahnung von forschungsintegrierender Lehre."*

DHBW Karlsruhe 2013/14
Methoden der Wirtschaftsinformatik

DHBW
Duale Hochschule
Baden-Württemberg
Karlsruhe

- Kontext
  - Interdiszipläneres Abschlussprojekt des Bachelorstudiengangs
  - Die 12 Studenten „gründen“ zwei Firmen, stellen ein Angebot und entwickeln konkurrierende Lösungen
  - Einführung in TDD und C4J (Trainingsdauer: 2 Tage)
- Rahmenbedingungen
  - Aufwandsrahmen: Das Projekt musste mit einem Aufwand 150 PT (Personentage) pro Team umgesetzt werden.
  - Der zeitliche Rahmen: November 2013 bis Mai 2014 (mit einer drei-monatigen Unterbrechung für die Bachelorarbeit)
  - Agiles Vorgehen, d.h. der Kunde agiert als Product Owner und priorisiert und verfeinert die Anforderung just-in-time, Sprint für Sprint

Aufgabe: Entwickeln eines Informations- und Navigationsportals für die DHBW Karlsruhe

DHBW
Duale Hochschule
Baden-Württemberg

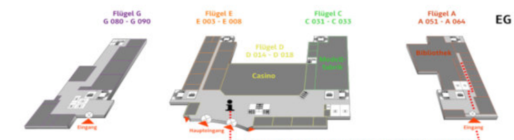
Room: A051

May 20, 2014 2:50:29 PM

Information

Typ: Planspiellabor

Link to Rapla: 1. Link to Rapla



Open this page on your Smartphone. Scan this QR-code.

Languages Back Home

DHBW Karlsruhe 2013/14
Team 1 : StudiInf

DHBW
Duale Hochschule
Baden-Württemberg
Karlsruhe

- Verwendete Technologie
  - Java
  - GWT als Web Application Framework

- Zitate aus dem Erfahrungsbericht des Teams
  - *„Nach dem erfolgreichen Abschließen des Workshops war der Eindruck innerhalb der Gruppe durchgehend positiv.“*
  - *„Zudem waren wir überrascht, dass diese Vorgehensweise bisher so unbekannt ist.“*
  - *„Die Rücksprache ergab, dass eine Unterstützung von GWT zwar generell möglich, aber nicht im Zeitraum des Projektes umsetzbar sei.“*
  - *„In Anbetracht der limitierten Zeit, haben wir uns daher gegen die Nutzung von C4J entschieden, obwohl wir grundsätzlich viele Vorteile in C4J und TbC sehen.“*

DHBW Karlsruhe 2013/14
Team 1 : StudiInf

DHBW
Duale Hochschule
Baden-Württemberg
Karlsruhe

- *„Dadurch mussten wir uns mehrfach in den Quelltext der einzelnen Funktionalitäten einarbeiten. Zusätzlich konnten wir nicht von der Schutzfunktion automatisierter Tests profitieren. Somit wurde ein höherer manueller Testaufwand erforderlich.“*
- *„Soweit technisch möglich würden wir TbC gerne für nachfolgende Projekte nutzen.“*

- *Hinweis: TbC = Testing by Contract, d.h. das Einführen von Contracts in Kombination mit Test-Bots (Ansatz von Jonas Bergström)*

DHBW Karlsruhe 2013/14
Team 2 : Pathfinder

DHBW Duale Hochschule Baden-Württemberg Karlsruhe

- Verwendete Technologie
  - Java
  - Vaadin als Web Application Framework

- Zitate aus dem Erfahrungsbericht des Teams
  - *„Da vor dem Seminar keiner von uns Erfahrungen mit TDD oder gar C4J hatte, waren wir anfangs skeptisch: Wie soll man einen Test schreiben, für eine Klasse, die gar nicht existiert und warum sollte man sich die Mühe machen und mit viel Zeitaufwand einen Vertrag für eine Klasse zu erstellen?“*
  - *„Zu Beginn verfielen wir schnell in unsere alte Programmierweise ohne den Einsatz von TDD und C4J, da wir uns dachten, damit schneller voranzukommen.“*
  - *„Erst als die Fehler überhand nahmen und den Erfolg des Projektes gefährdeten, schwenkten wir auf die neu erlernte Vorgehensweise um.“*

DHBW Karlsruhe 2013/14
Team 2 : Pathfinder

DHBW
Duale Hochschule
Baden-Württemberg
Karlsruhe

- *„Bald waren wir begeistert von der Einfachheit, mit der in C4J Verträge erstellt werden können und wie flexibel diese sich innerhalb des TDD-Zyklus anpassen lassen."*
- *„Doch am meisten überzeugt hat uns, wie schnell Fehler im Code erkannt werden konnten und wie präzise die jeweilige Fehlerursache ermittelt werden konnte."*
- *„Tappten wir bei unserer ursprünglichen Programmierweise ohne TDD und C4J oft im Dunkeln, machte sich bei einer Verletzung einer der Vor- oder Nachbedingungen sofort der Contract bemerkbar und lieferte präzise Angaben über die Ursache."*

DHBW Karlsruhe 2013/14
Team 2 : Pathfinder

DHBW
Duale Hochschule
Baden-Württemberg
Karlsruhe

- *„Durch dieses Vorgehen konnten wir die Anzahl der Fehler schnell minimieren und die weitere Entwicklung am Projekt ging insgesamt schneller voran, obwohl wir viel Zeit für die Erstellung der Verträge aufwenden mussten.“*
- *„Der Nachteil des hohen Zeitaufwands für die Erstellung der Contracts wird durch eine kürzere Entwicklungszeit wettgemacht, die sich aus der besseren Wartbarkeit des Codes und einer schnelleren Fehlerbehebung ergibt.“*
- *„Wir sind uns einig, dass wir in zukünftigen Projekten TDD und C4J wieder einsetzen werden.“*

Inhaltsübersicht
TDD with Contracts

1. Prolog
2. Was ist TDD with Contracts?
3. Wie funktioniert TDD with Contracts (Demo)?
4. Welche Erfahrungen gibt es mit TDD with Contracts?
5. Epilog
6. Fragen & Antworten

http://c4j.vksi.de

VKSI

Overview
Ease of use
Powerful
Six principles
Examples
Syntax reference
Inheritance of contracts
Installation
Configuration
Guidelines

Download C4J 6.0.0
C4J Eclipse Plugin

Download C4J 2.7.5

Overview

Contracts for Java (C4J) is a *Contracts* (from Design by Contract, see Wikipedia DbC definition) framework for Java 1.6 and later. The primary goal for C4J is ease of use. Contracts are about design and quality, aspects of programming that a lot programmers don't spend enough time and energy on.
Therefore a Contracts framework must be simple and painless to use. At the same time the framework must be powerful.
C4J is simple *and* powerful.

We are not going to try to convince any readers that Contracts are indeed a very powerful design and quality assurance technique, so if you are not already convinced of that, please follow the links above and you may be convinced to try this tool out! These are our favorites though:

- *Contracts* consist of *preconditions* and *postconditions*, which are systematically defined method by method.
- The *class invariant* ensures the *DRY principle (Don't Repeat Yourself)* for contracts by defining all those assertions, which must be fulfilled at any visible state of an object, only once.
- Contracts can be linked to classes and interfaces.
- Contracts are inherited by extending a class or by implementing an interface which is guarded by a contract.
- To be able to define meaningful contracts you are forced to split lengthy methods into small, well defined, methods with a single responsibility.
- As your contracts are checked at runtime, your classes are validated against the *real* usage of your application, not against some test cases that may not even be real use cases.
- If you are dealing with legacy code that you are afraid of refactoring, *external* contracts are perfect to add to existing code with no risk involved.

finance

Interface AccountSpec

deposit

```
void deposit(int amount)
```

Precondition

```
amount > 0 : "amount > 0"
```

Postcondition

```
getBalance() == old(getBalance()) + amount : "balance = old balance + amount"
```

---

withdraw

```
void withdraw(int amount)
```

Precondition

```
amount > 0 : "amount > 0"
getBalance() >= amount : "balance >= amount"
```

Postcondition

```
getBalance() == old(getBalance()) - amount : "balance = old balance - amount"
```

---

getBalance

```
@Pure
int getBalance()
```

Postcondition

```
result >= 0 : "result >= 0"
```

Clean Contract: Wie formuliert man einen vollständigen Vertrag?
Die 6 Prinzipien der Vertragserstellung

Prinzip 6
Formuliere Invarianten
um die unveränderlichen Eigenschaften des Objekts zu beschreiben. Wähle diejenigen Attribute, die den Anwender unterstützen, das konzeptionelle Modell der Klasse zu verstehen.

Prinzip 5
Prüfe für jede Abfrage und jedes Kommando,
ob eine Vorbedingung erforderlich ist.

Prinzip 4
Erstelle für jedes Kommando eine Nachbedingung,
die den Wert jeder elementaren Abfrage auflistet.
Der vollständige sichtbare Effekt eines jeden Kommandos ist nun bekannt.

Prinzip 3
Erstelle für jede abgeleitete Abfrage eine Nachbedingung,
die beschreibt, welches Ergebnis zurückgegeben wird,
ausgedrückt durch eine oder mehrere elementare Abfragen.

Prinzip 2
Trenne elementare Abfragen von abgeleiteten Abfragen.
Abgeleitete Abfragen können durch elementare Abfragen ausgedrückt werden.

Prinzip 1
Trenne Abfragen durch `@Pure` von Kommandos.
Abfragen geben ein Ergebnis zurück, ändern jedoch nicht die sichtbaren Eigenschaften des Objekts.
Kommandos können das Objekt verändern, geben jedoch kein Ergebnis zurück.

In enger Anlehnung an: R. Mitchell / J. McKim: Design by Contract, by Example

Anmeldung zum C4J-Tutorial
www.vksi.de | Veranstaltungen

Home Login Kontakt Impressum

Software means Engineering

Home | Der Verein | Sneak Preview | Veranstaltungen | Newsletter | Magazin | BOGY-Praktikum | C4J

Veranstaltungen ▸ 13.06.2014 - TDD with Contracts

GO!

Online Anmeldung

Tutorial "TDD with Contracts" am 13.06.2014

Kursthema:
TDD with Contracts

WebSites zum Kursthema:
C4J-WebSite - http://c4j-team.github.com/C4J
C4J-Plugin - https://github.com/C4J-Team/C4J-Eclipse-Plugin/raw/master/update-site/

http://c4j-team.github.com/C4J