

Modul 3

Techniken

DI Gerhard Fließ

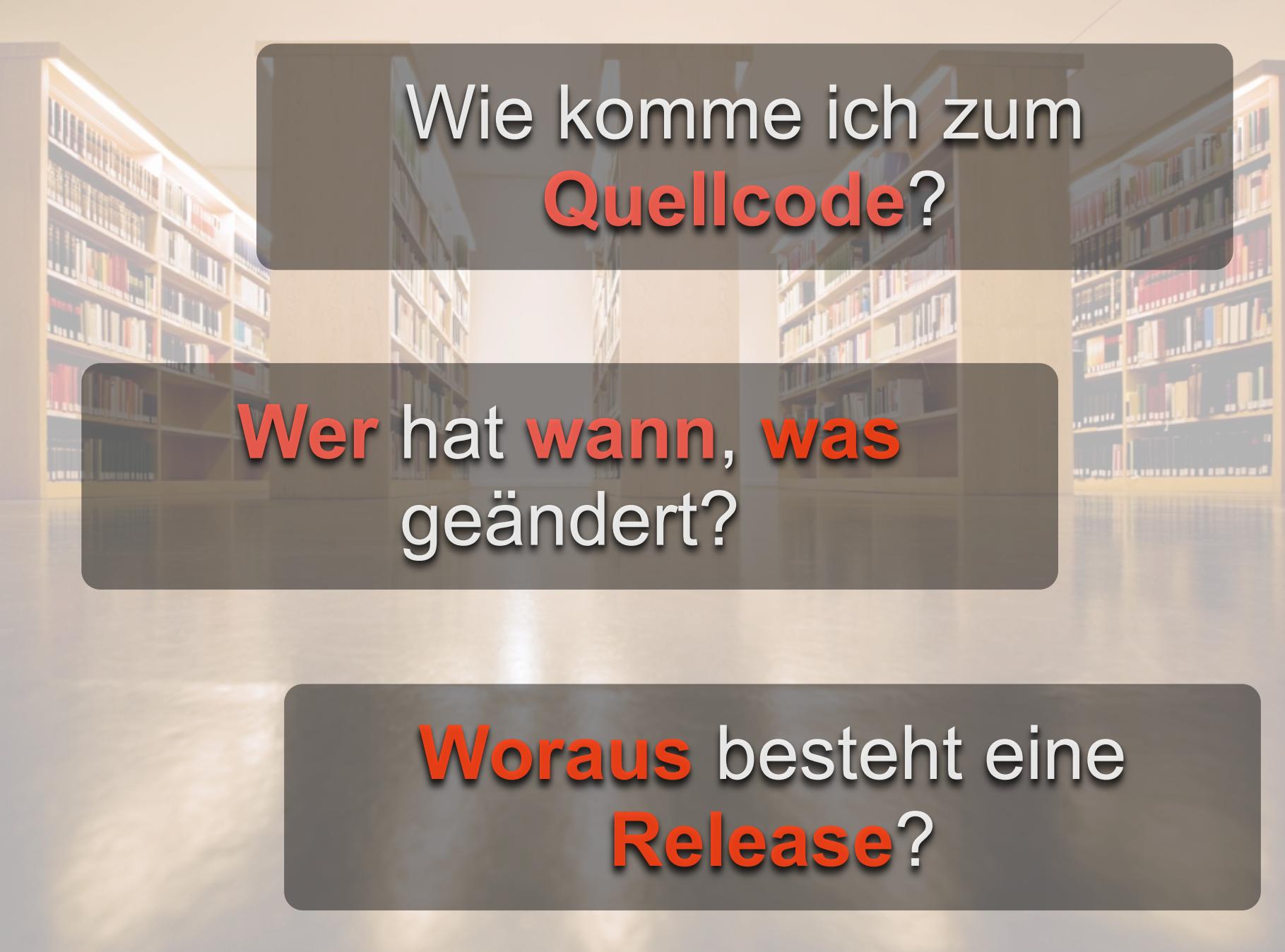
2013-04-16

Rückblick Modul 2





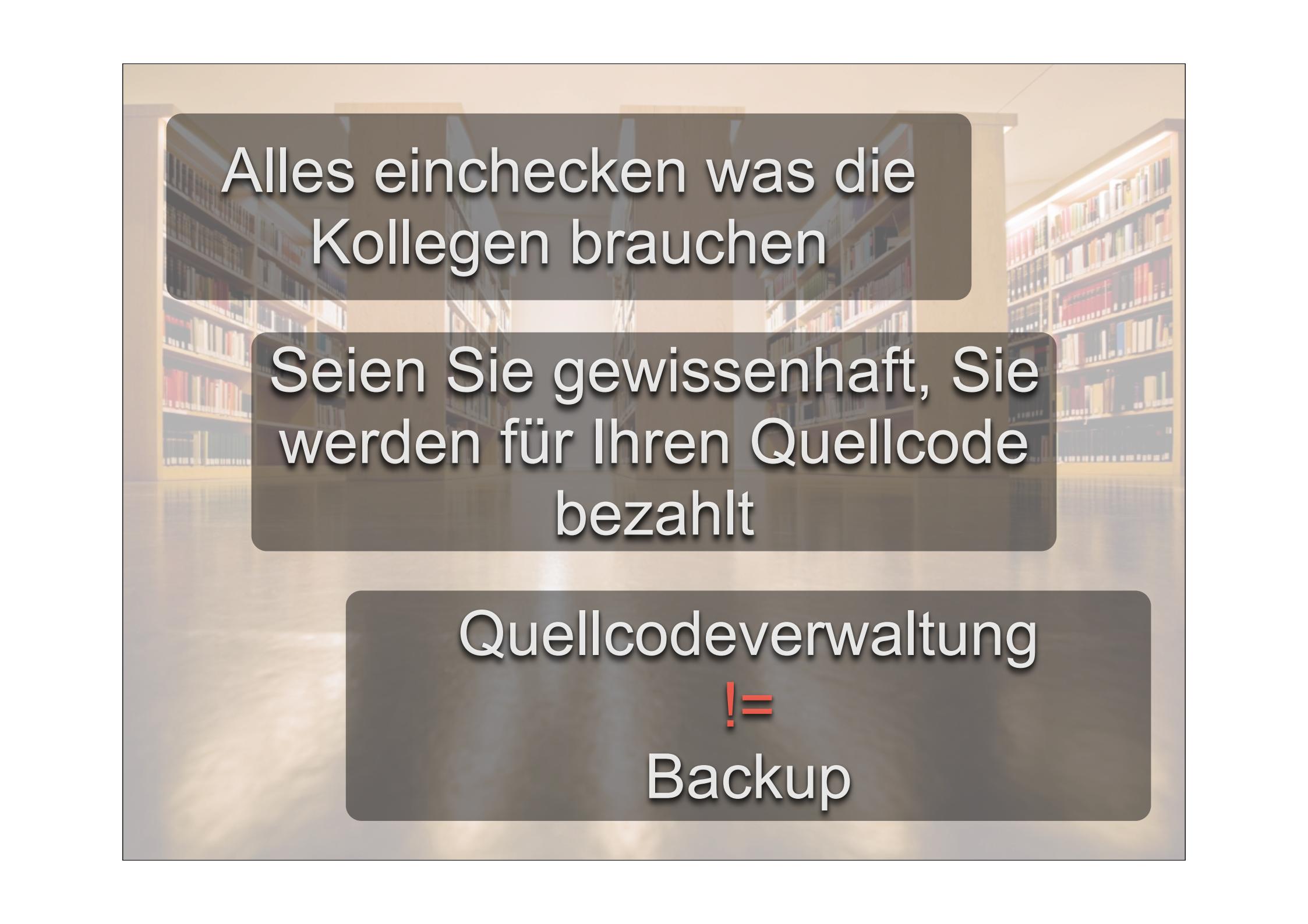
Quellcodeverwaltung ist
die *Bibliothek* jeder
Softwareentwicklung



Wie komme ich zum
Quellcode?

Wer hat **wann, was**
geändert?

Woraus besteht eine
Release?



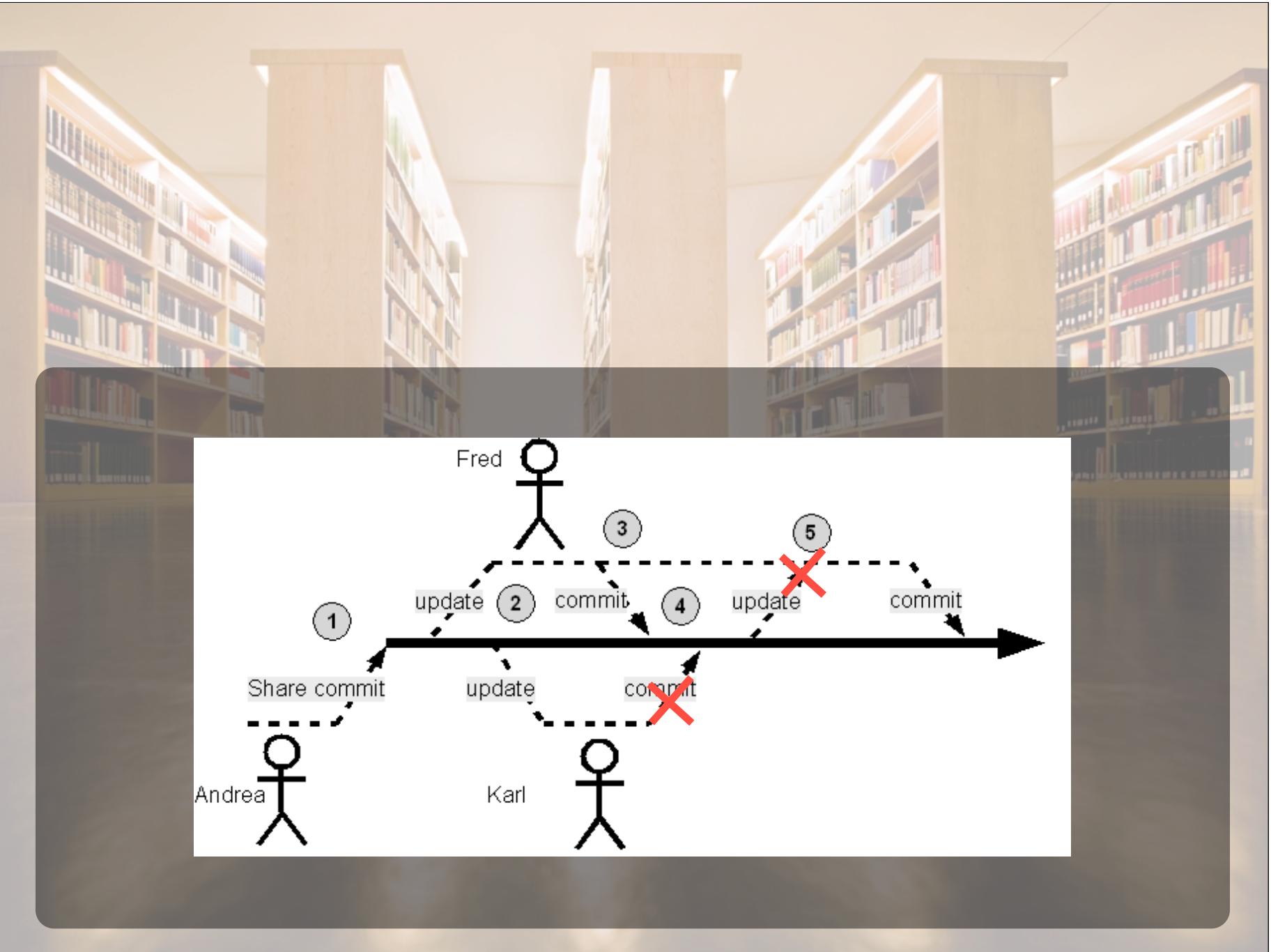
Alles einchecken was die
Kollegen brauchen

Seien Sie gewissenhaft, Sie
werden für Ihren Quellcode
bezahlt

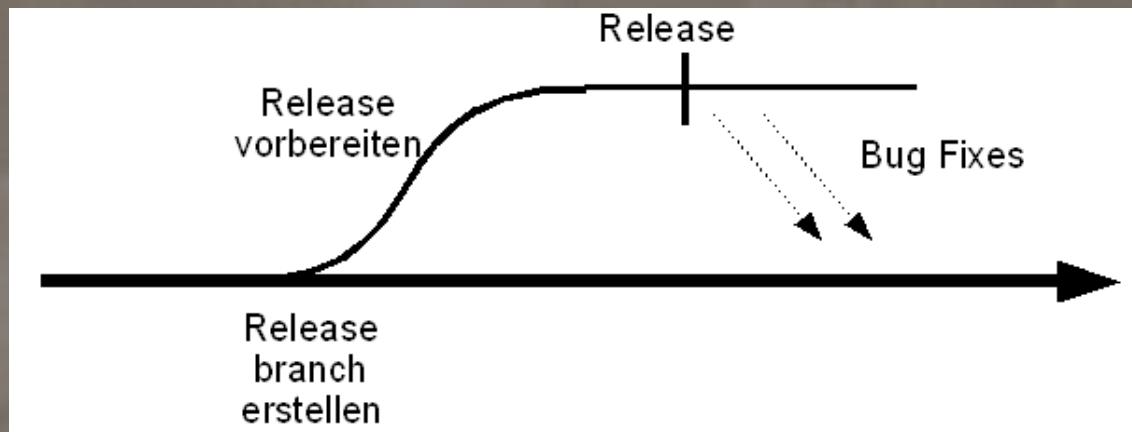
Quellcodeverwaltung

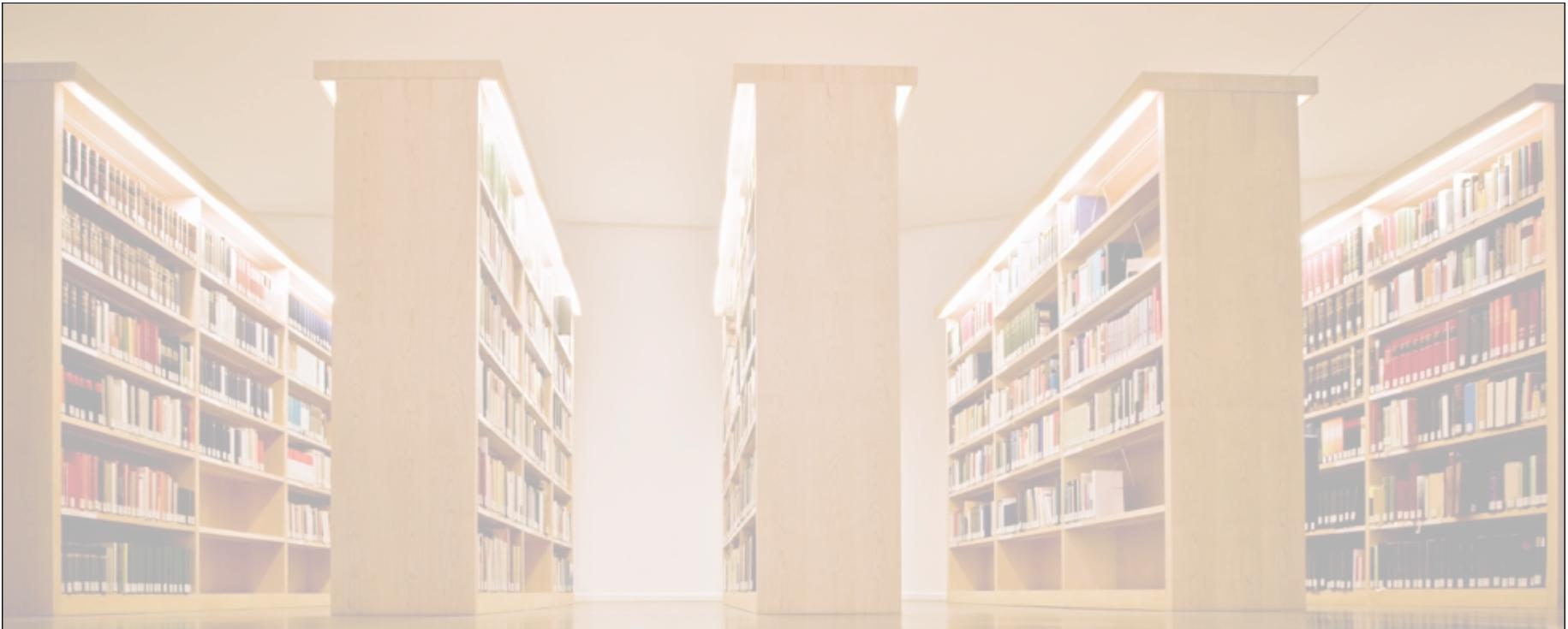
!=

Backup



Tags und Branches

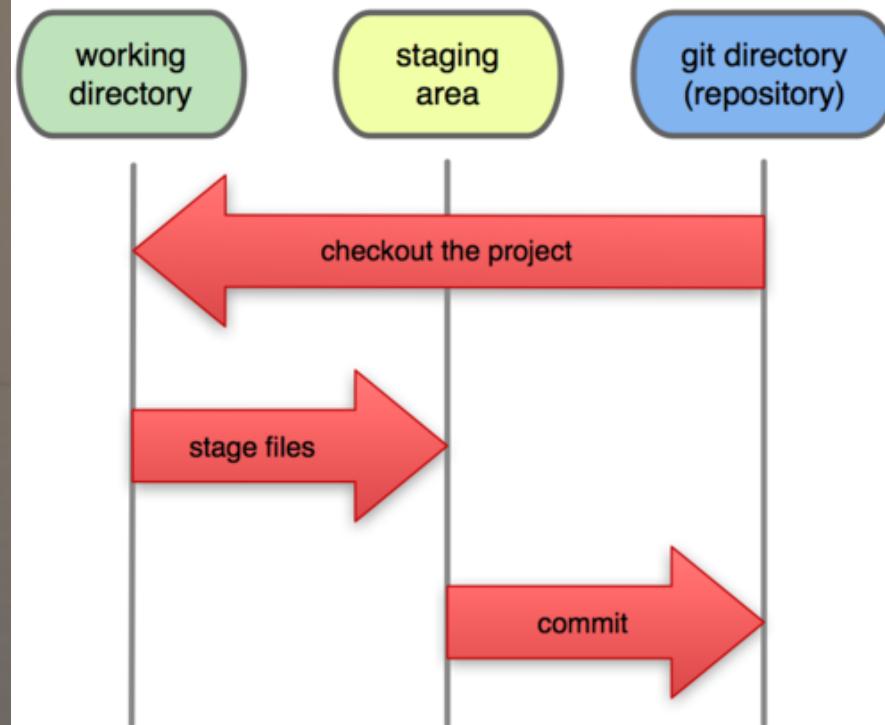




**Wie nenne ich die TAGS?
Keine Angst von Branches?
Trotzdem nicht für alles einen
Branch...**

GIT

Local Operations



<http://git-scm.com/book/en/Getting-Started-Git-Basics>

Git Cheat sheet

[http://rogerdudler.github.io/git-guide/files/
git_cheat_sheet.pdf](http://rogerdudler.github.io/git-guide/files/git_cheat_sheet.pdf)

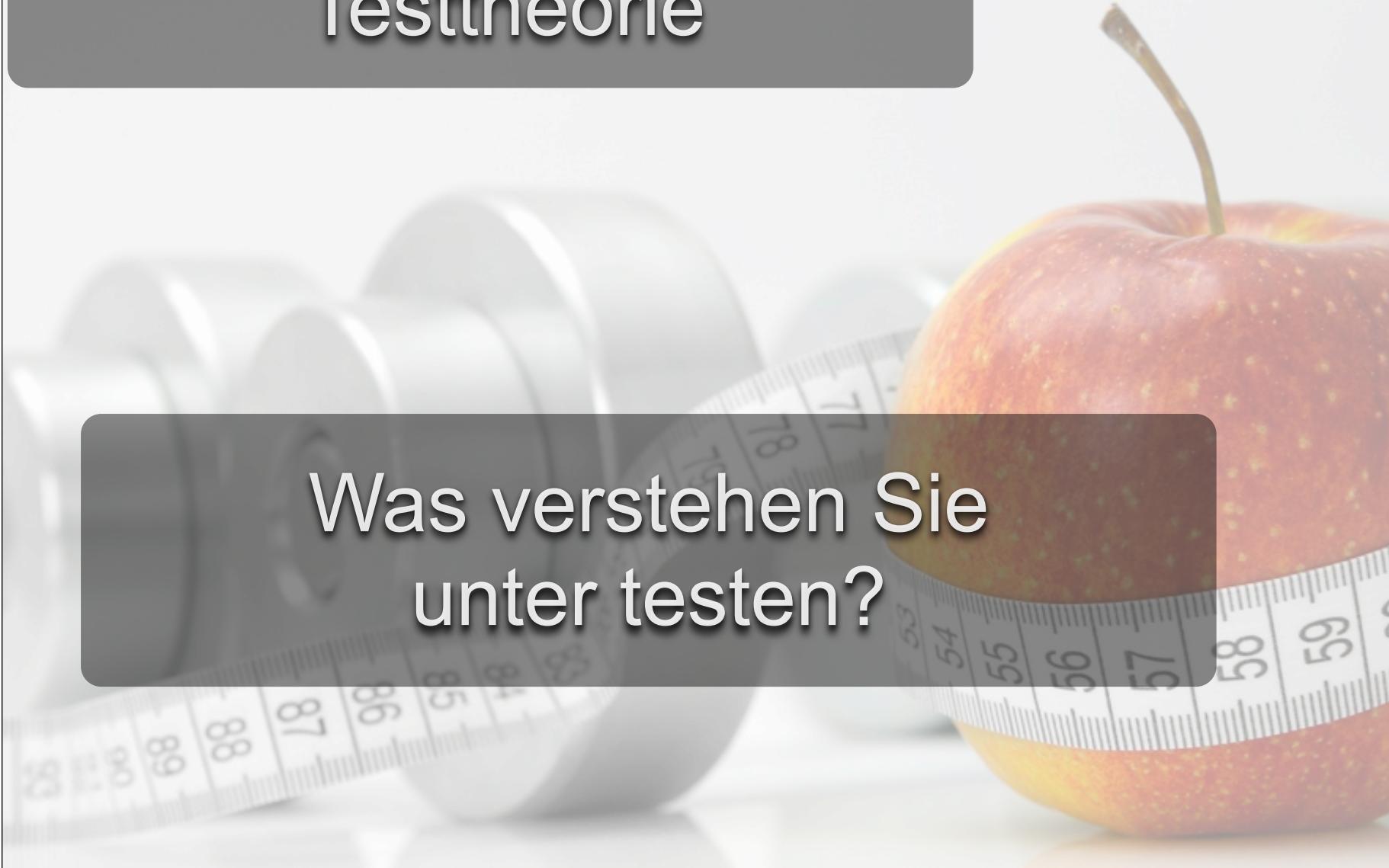
**Testen ist wie „Obst-Essen“
und „Sport-Betreiben“...**

**... alle halten es für
wichtig, aber
gemacht wird es zu
wenig.**

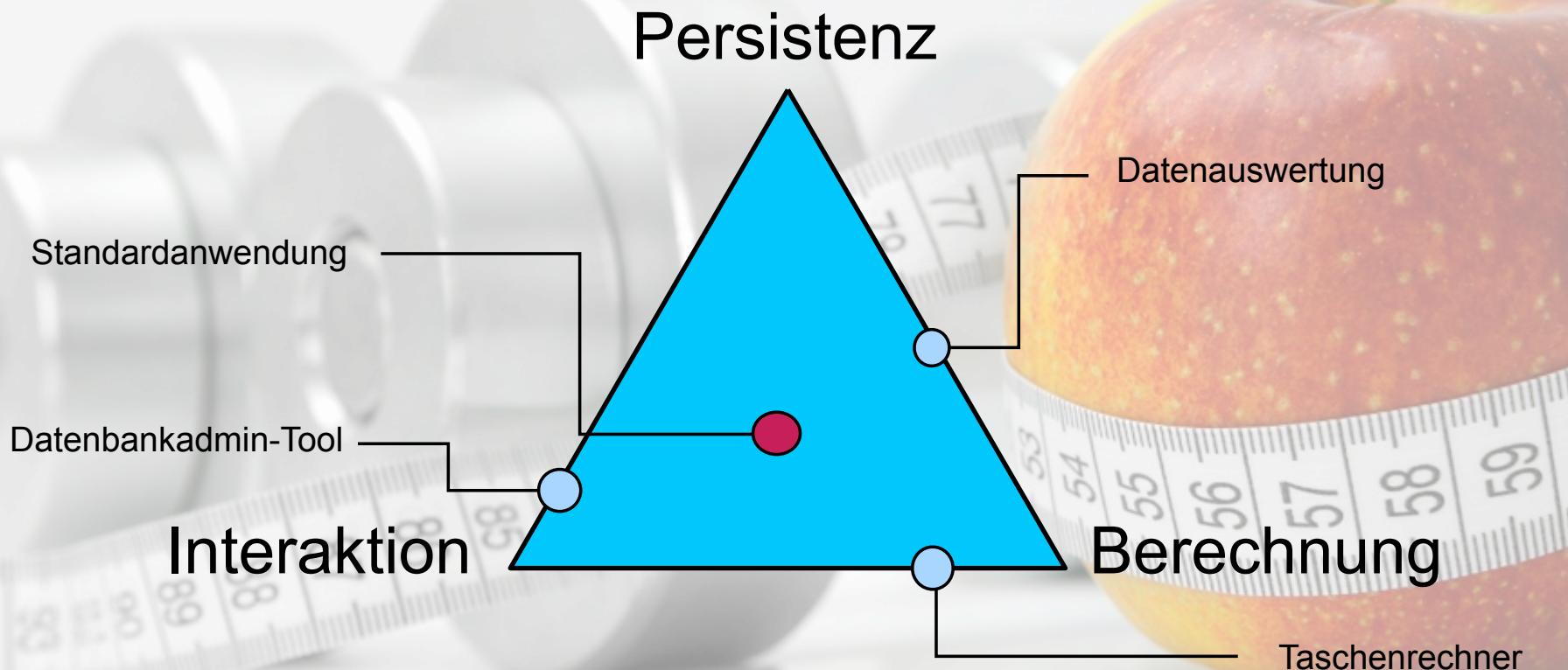


Testtheorie

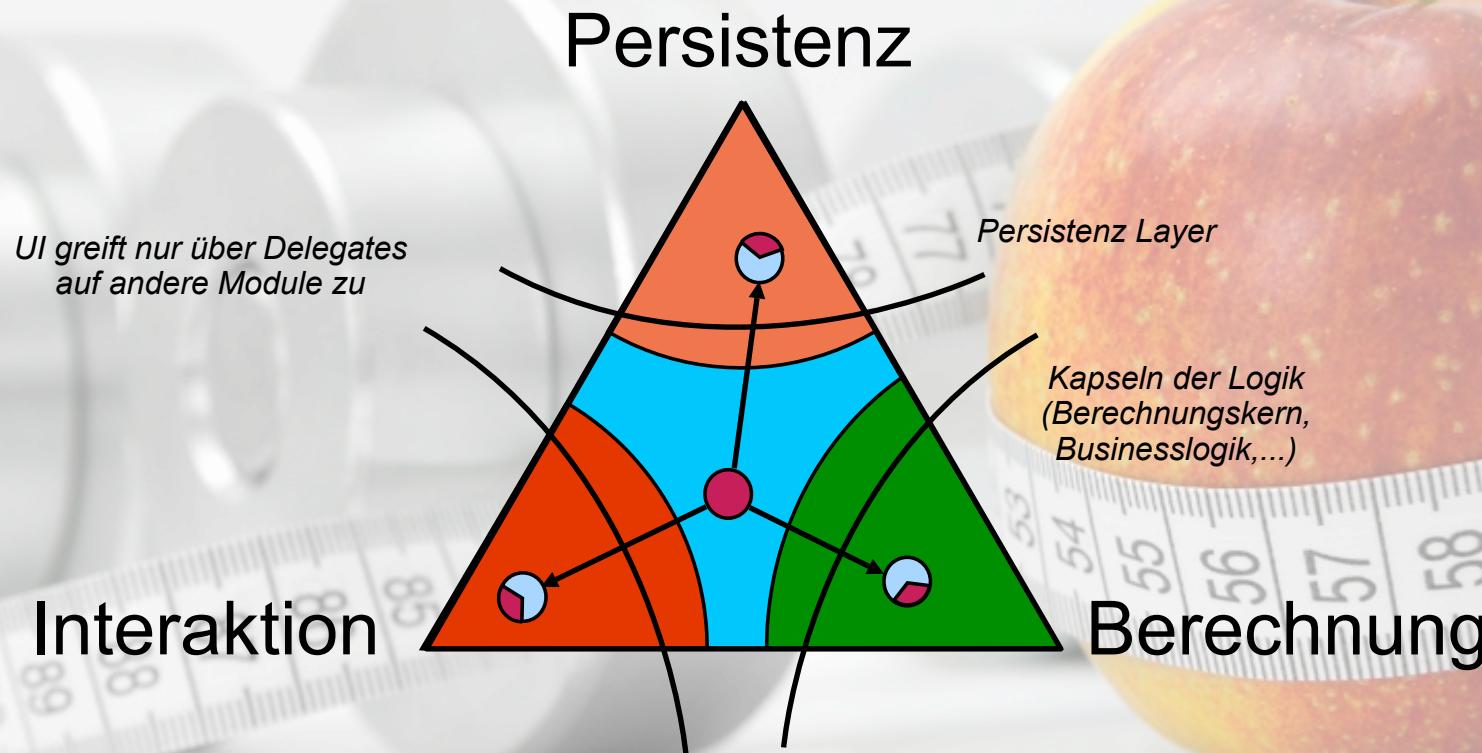
Was verstehen Sie
unter testen?



Aufgabenbereiche innerhalb einer Anwendung



Testbarkeit der Aufgabenbereiche



Integrations vs. Unit Tests



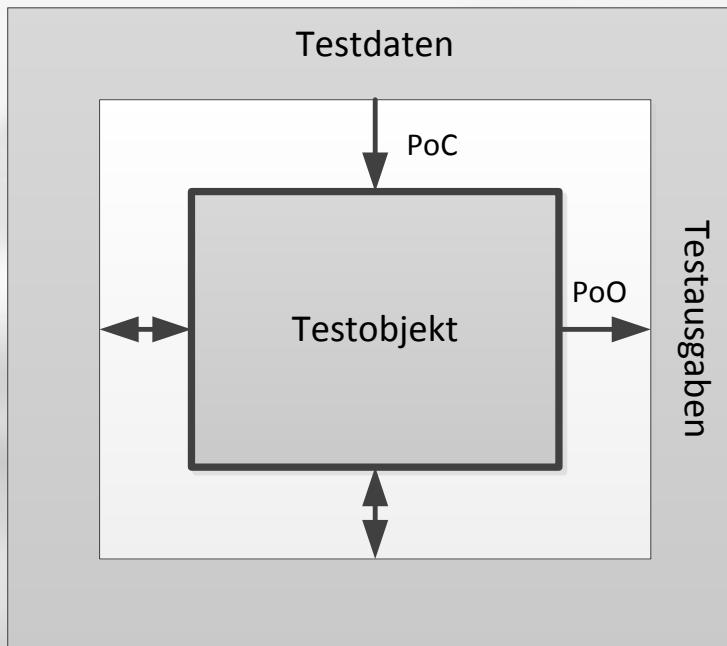


Positiv Testfälle testen
die korrekte Funktion

Negativ Testfälle testen
die Robustheit

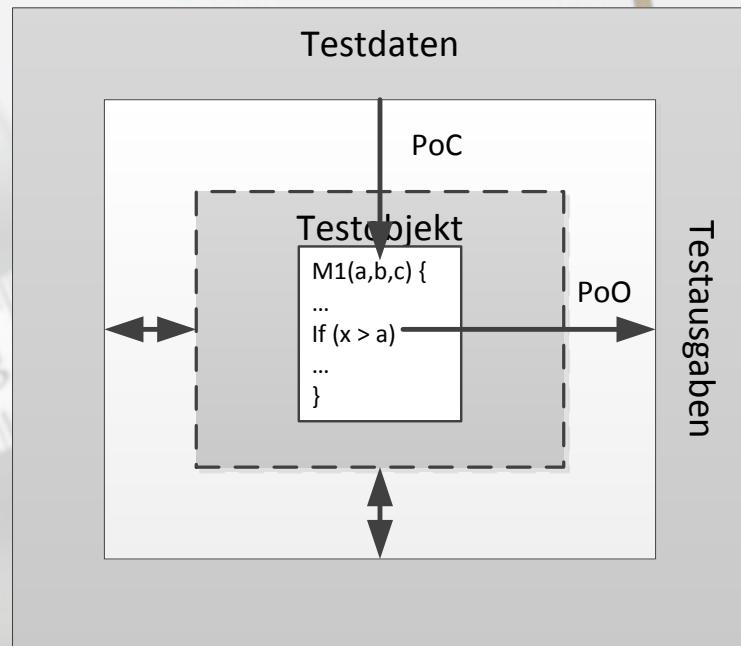
Whitbox / Blackbox

Black-Box-Verfahren



PoC und PoO „außerhalb“
des Testobjektes

White-Box-Verfahren



PoC und/oder PoO „innerhalb“
des Testobjektes

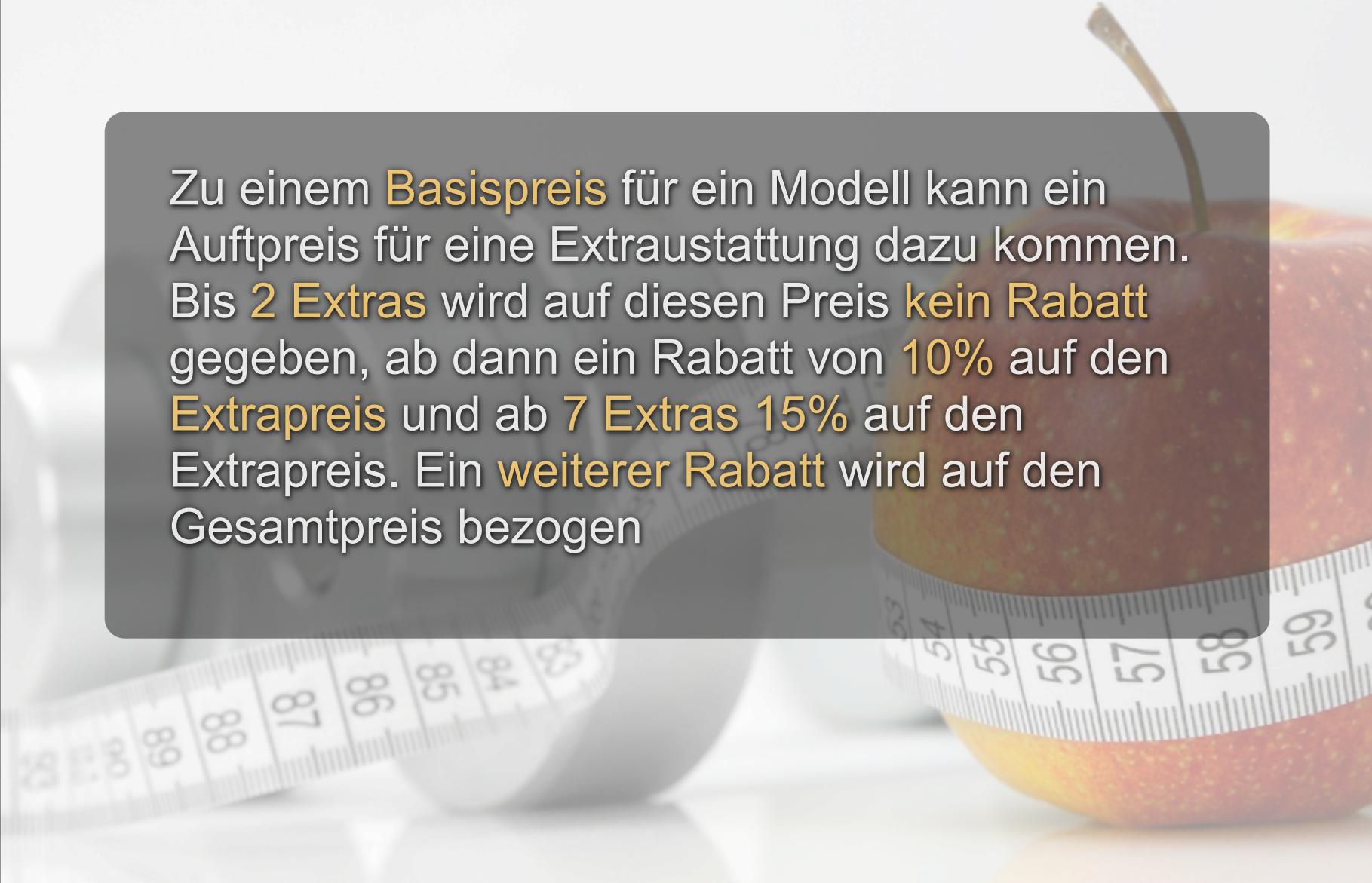
Äquivalenzklassen

Unterteilen des möglichen Wertebereichs in Teile (=Äquivalenzklasse) bei denen sich das Testobjekt gleich verhält. Gültige und ungültige Daten!

Anzahl der positiv-Testfälle Produkt der Anzahl der einzelnen gültigen ÄK
Anzahl der negativ-Testfälle ist Summe der Anzahl der einzelnen ungültigen ÄK

Beispiel

Zu einem **Basispreis** für ein Modell kann ein Auftpreis für eine Extraustattung dazu kommen. Bis **2 Extras** wird auf diesen Preis **kein Rabatt** gegeben, ab dann ein Rabatt von **10%** auf den **Extrapreis** und ab **7 Extras** **15%** auf den Extrapreis. Ein **weiterer Rabatt** wird auf den Gesamtpreis bezogen



```
double calcPreis(double  
basispreis, double extrapreis,  
int extras, double rabatt)
```

Parameter	Bezeichner	Äquivalenzklasse	Beispiel
Basispreis	g11	[0., ... MAX_DOUBLE]	2000
	u11	[Min_Double, ... ,0[-2.0
	u12	NaN	abc
Extrapreis	g21	[0., ... MAX_DOUBLE]	500
	u21	[Min_Double, ... ,0[-2.0
	u22	NaN	abc
Extras	g31	[0., ... 2]	1
	g32	[3, ... 6]	4
	g33	[7, ... Max_Int]	8
	u31	[Min_Double, ... ,0[-2.0
	u32	NaN	abc
Rabatt	g41	[0., ... 100]	5
	u41	[Min_Double, ... ,0[-2.0
	u42]100, ... Max_Double]	120
	u43	NaN	abc

1*1*3*1=3 positiv Testfälle
 2+2+2+3=9 negativTestfälle

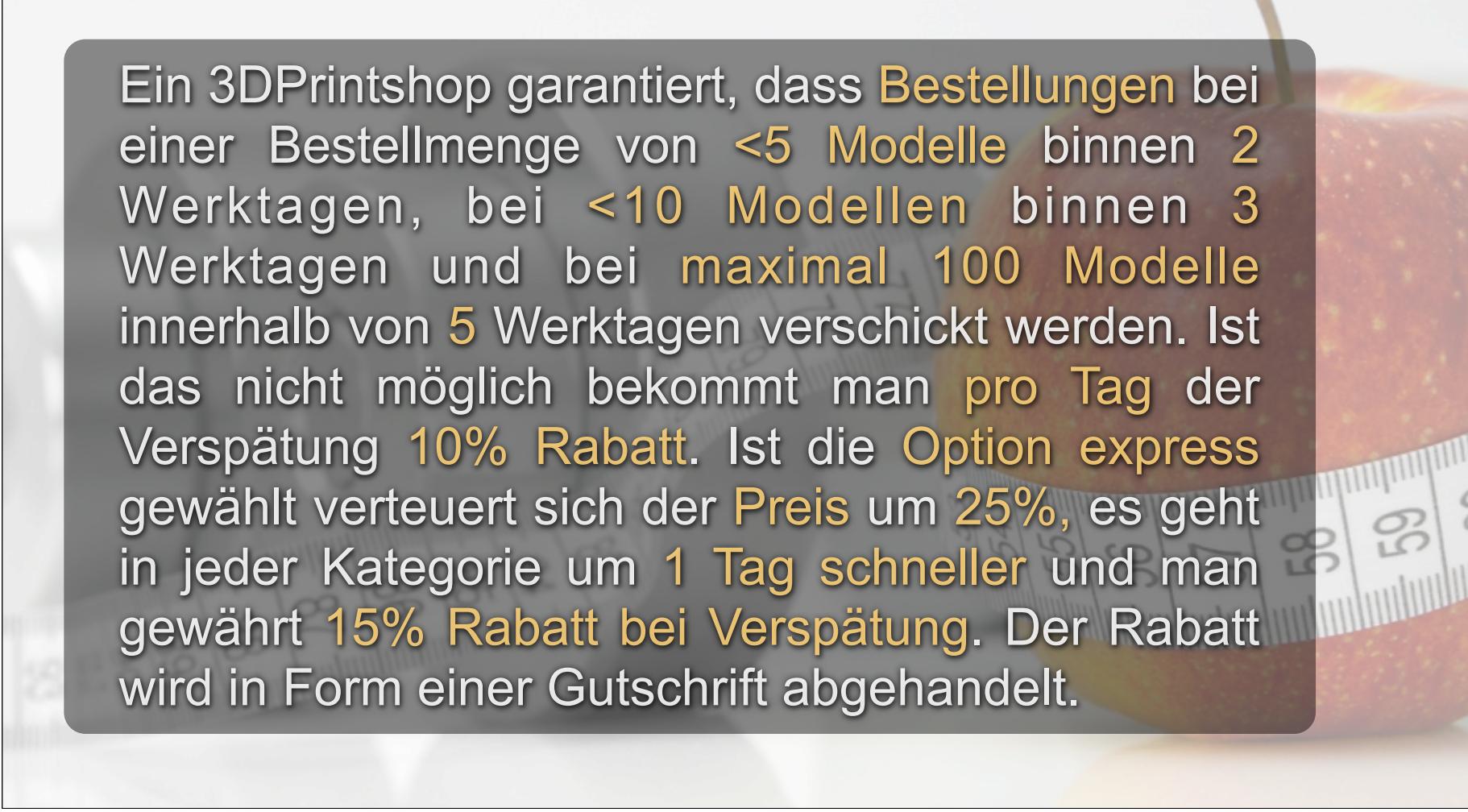


```
double calcPreis(double  
basispreis, double extrapreis,  
int extras, double rabatt)
```

Id	Basispreis	Extrapreis	Extras	Rabatt	Preis
ptf1	2000	500	1	10	2250
ptf2	2000	600	4	10	2286
ptf3	2000	800	8	10	2412
ntf1	-2	500	1	10	invalid
ntf2	abc	500	1	10	invalid
ntf3	2000	-2	1	10	invalid
ntf4	2000	abc	1	10	invalid
ntf5	2000	500	-4	10	invalid
ntf6	2000	500	abc	10	invalid
ntf7	2000	500	1	-2	invalid
ntf8	2000	500	1	120	invalid
ntf9	2000	500	1	abc	invalid

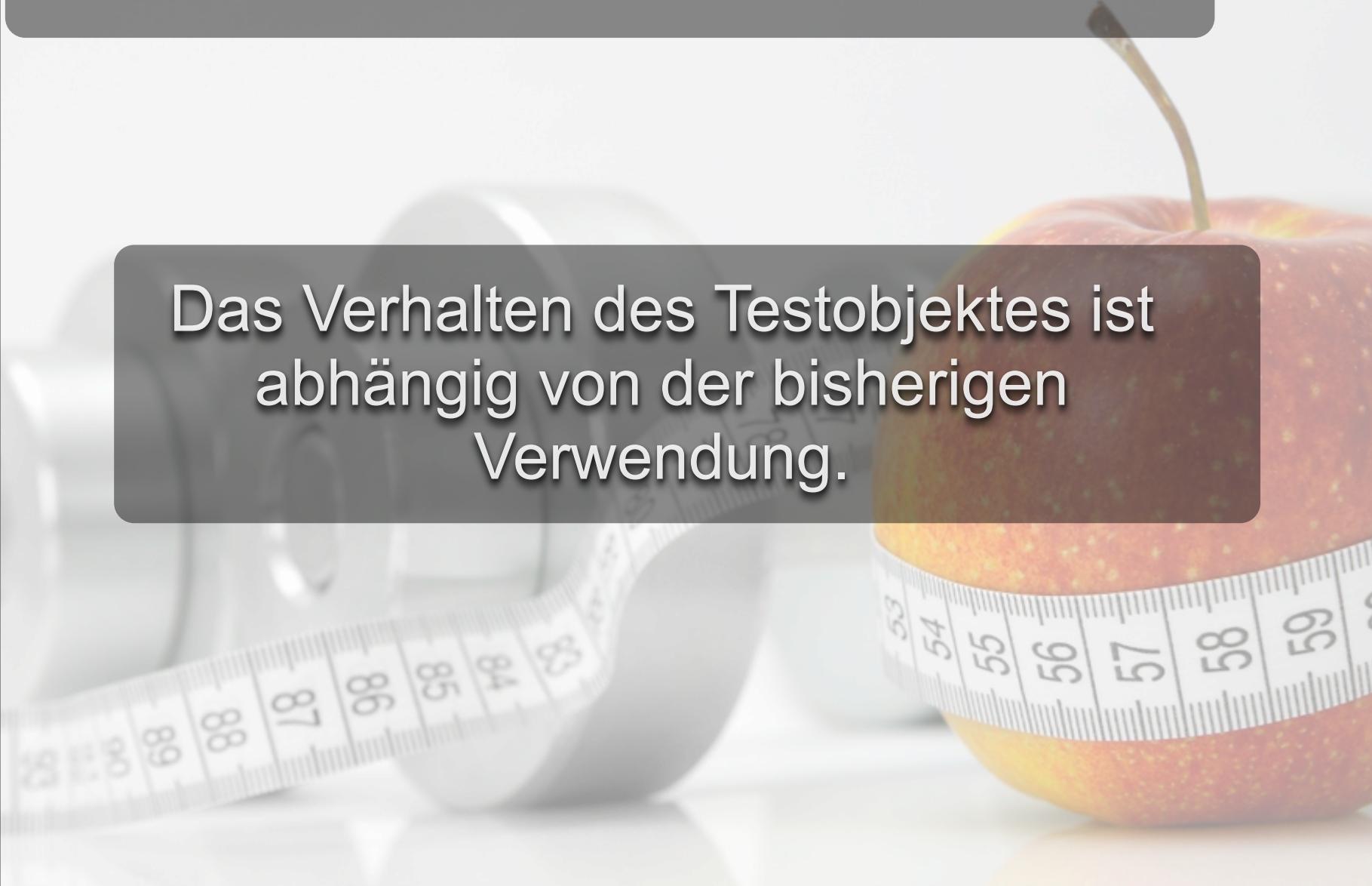
Übung zu Äquivalenzklassen

Ein 3DPrintshop garantiert, dass Bestellungen bei einer Bestellmenge von <5 Modelle binnen 2 Werktagen, bei <10 Modellen binnen 3 Werktagen und bei maximal 100 Modelle innerhalb von 5 Werktagen verschickt werden. Ist das nicht möglich bekommt man pro Tag der Verspätung 10% Rabatt. Ist die Option express gewählt verteuerst sich der Preis um 25%, es geht in jeder Kategorie um 1 Tag schneller und man gewährt 15% Rabatt bei Verspätung. Der Rabatt wird in Form einer Gutschrift abgehandelt.



Zustandsbasiertes Testen

Das Verhalten des Testobjektes ist
abhängig von der bisherigen
Verwendung.

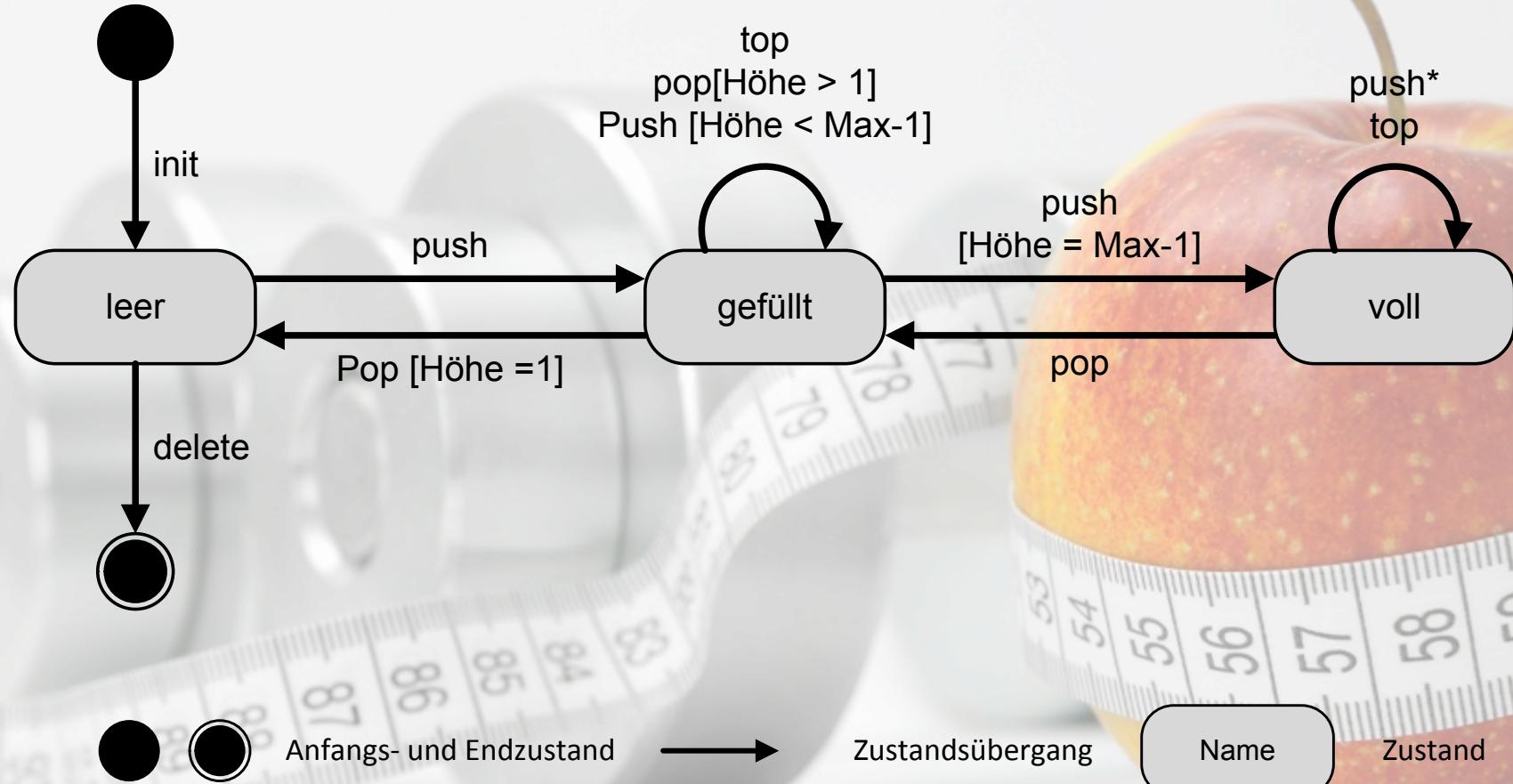


Zustandsbasiertes Testen

Erstellen eines Automaten der das Verhalten beschreibt.

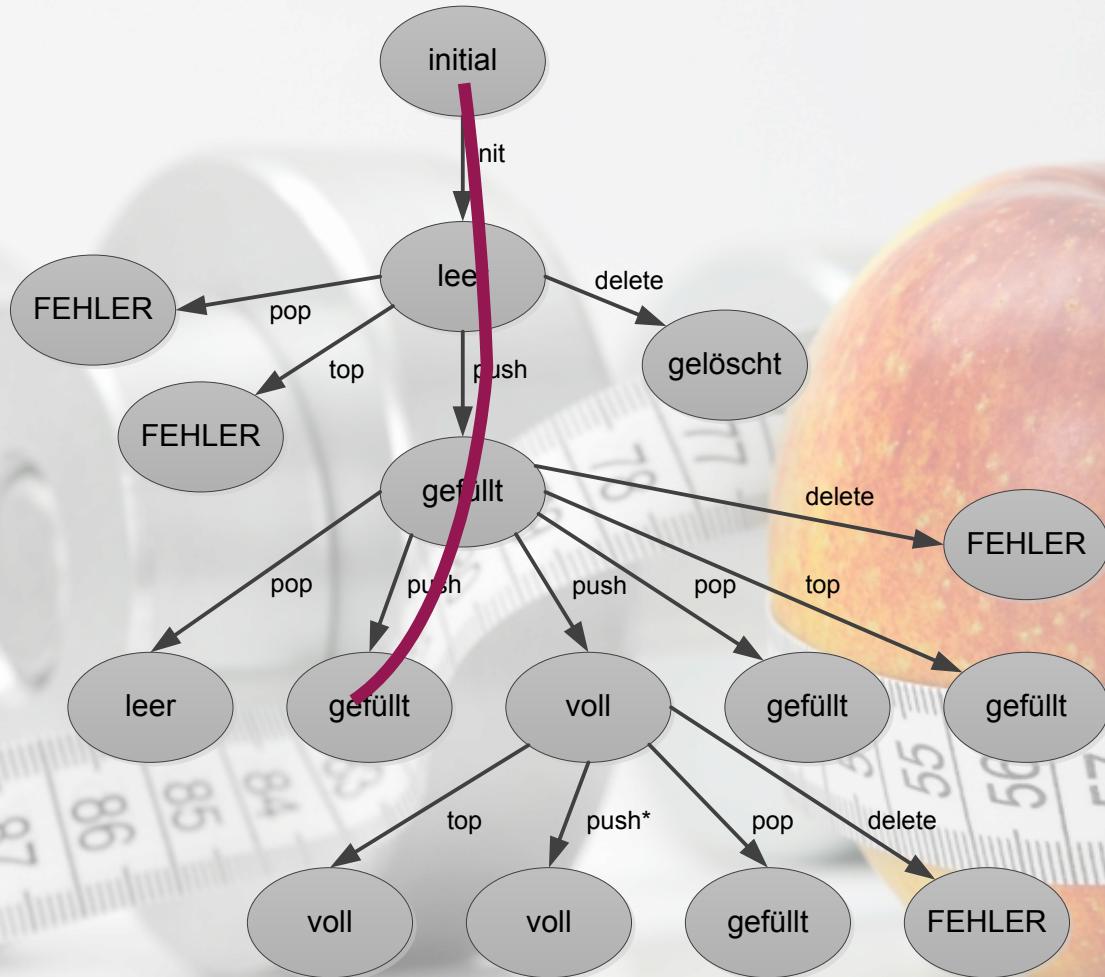
Erstellen eines Übergangsbaumes inkl fehlerhaften Aufrufen. Wege im Baum sind Aufrufsequenzen im Testfall

Zustandsbasiertes Testen



Beispiel eines Zustandsdiagramms (Spillner & Linz, 2005, S. 129)

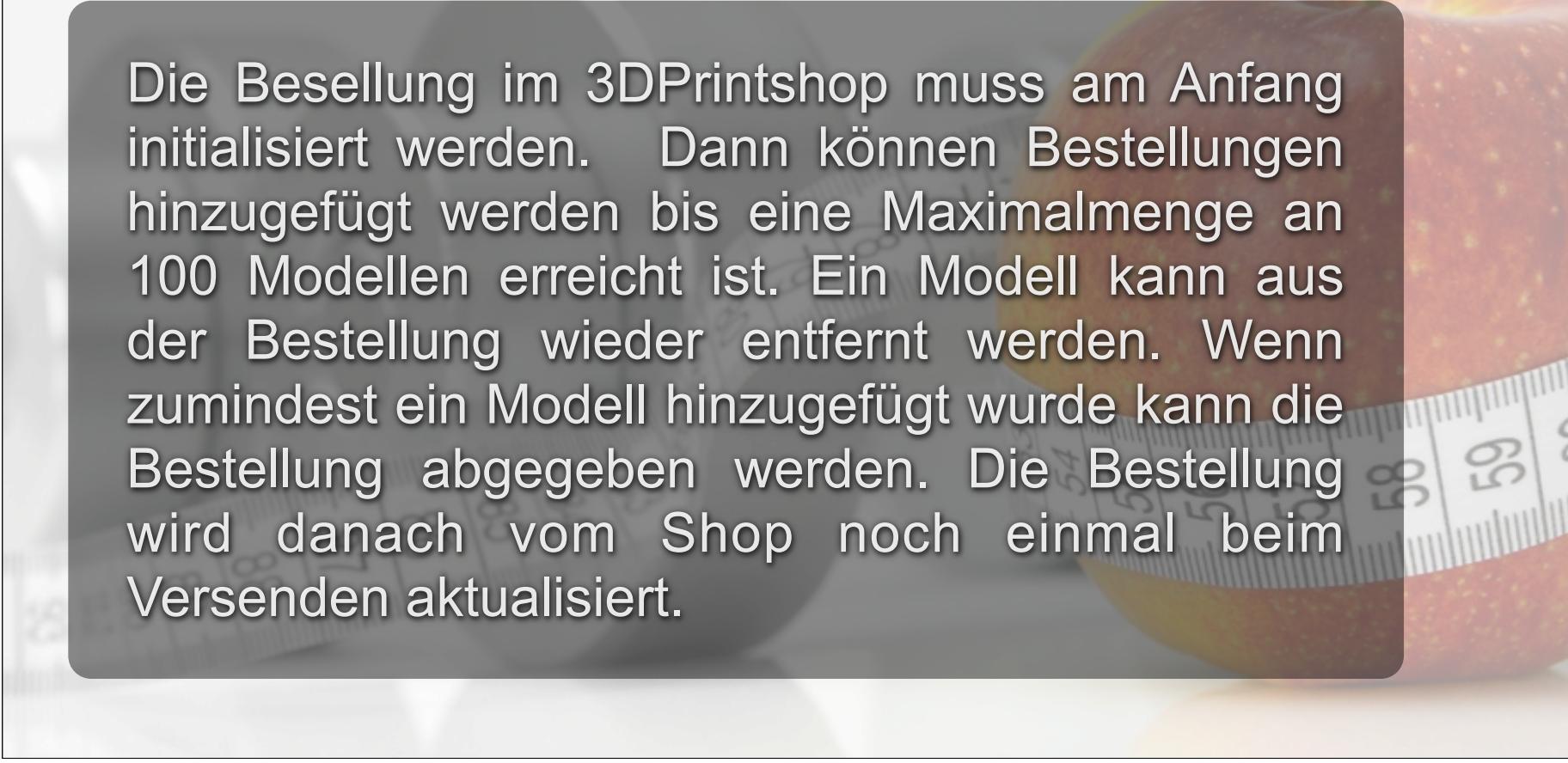
Zustandsbasiertes Testen

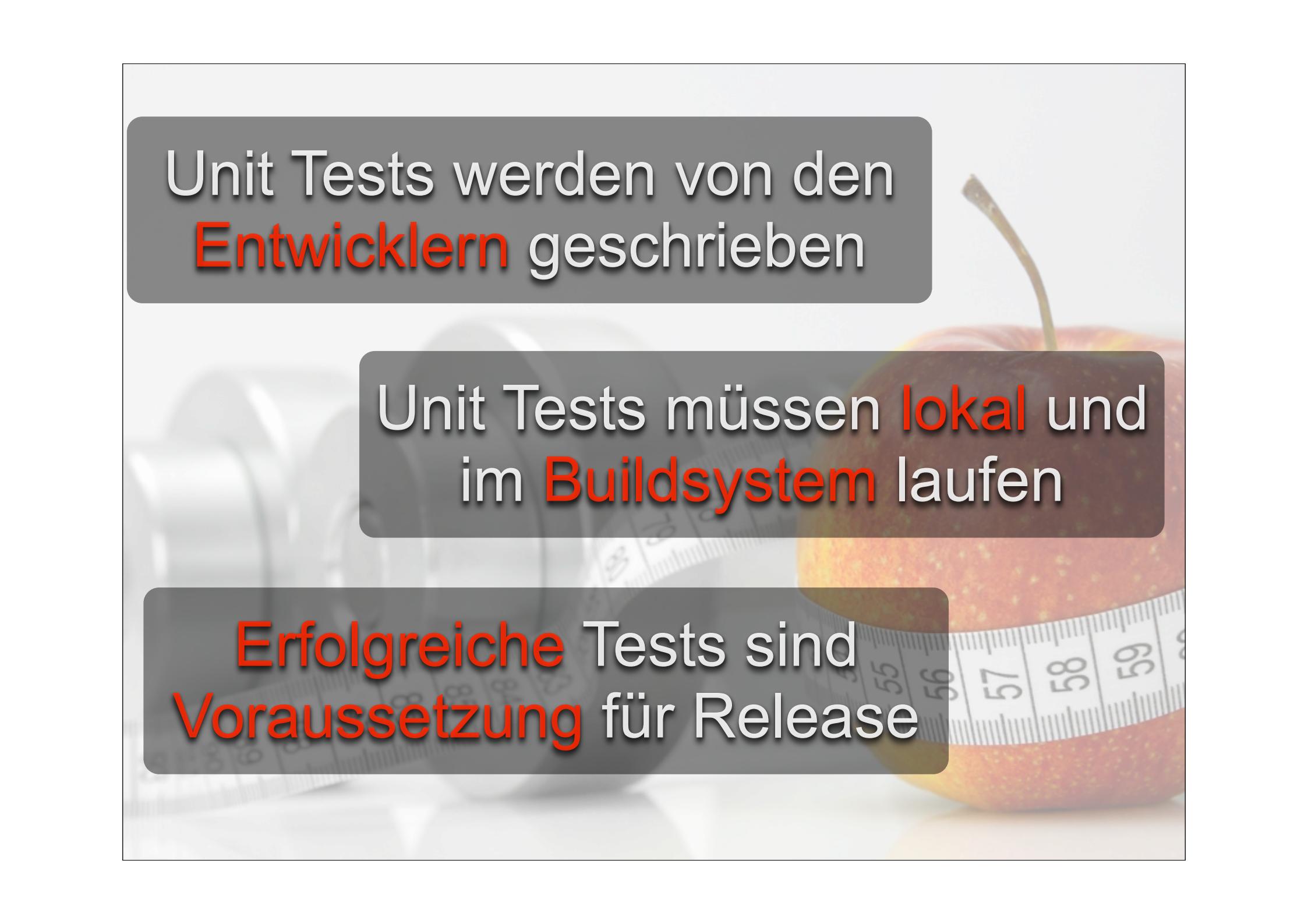


Beispiel eines Übergangsbaumes (Spillner & Linz, 2005, S. 132)

Übung zu Zustandsbasiertem Testen

Die Besellung im 3DPrintshop muss am Anfang initialisiert werden. Dann können Bestellungen hinzugefügt werden bis eine Maximalmenge an 100 Modellen erreicht ist. Ein Modell kann aus der Bestellung wieder entfernt werden. Wenn zumindest ein Modell hinzugefügt wurde kann die Bestellung abgegeben werden. Die Bestellung wird danach vom Shop noch einmal beim Versenden aktualisiert.





Unit Tests werden von den
Entwicklern geschrieben

Unit Tests müssen **lokal** und
im **Buildsystem** laufen

Erfolgreiche Tests sind
Voraussetzung für Release

Ein einfacher Rechner

1. Unterstützt einfache Rechenoperationen
2. Stack-Rechner
3. Einlesen der Rechnung aus einer XML Datei
4. Trennen von Parser und Logik
5. Im Team umgesetzt

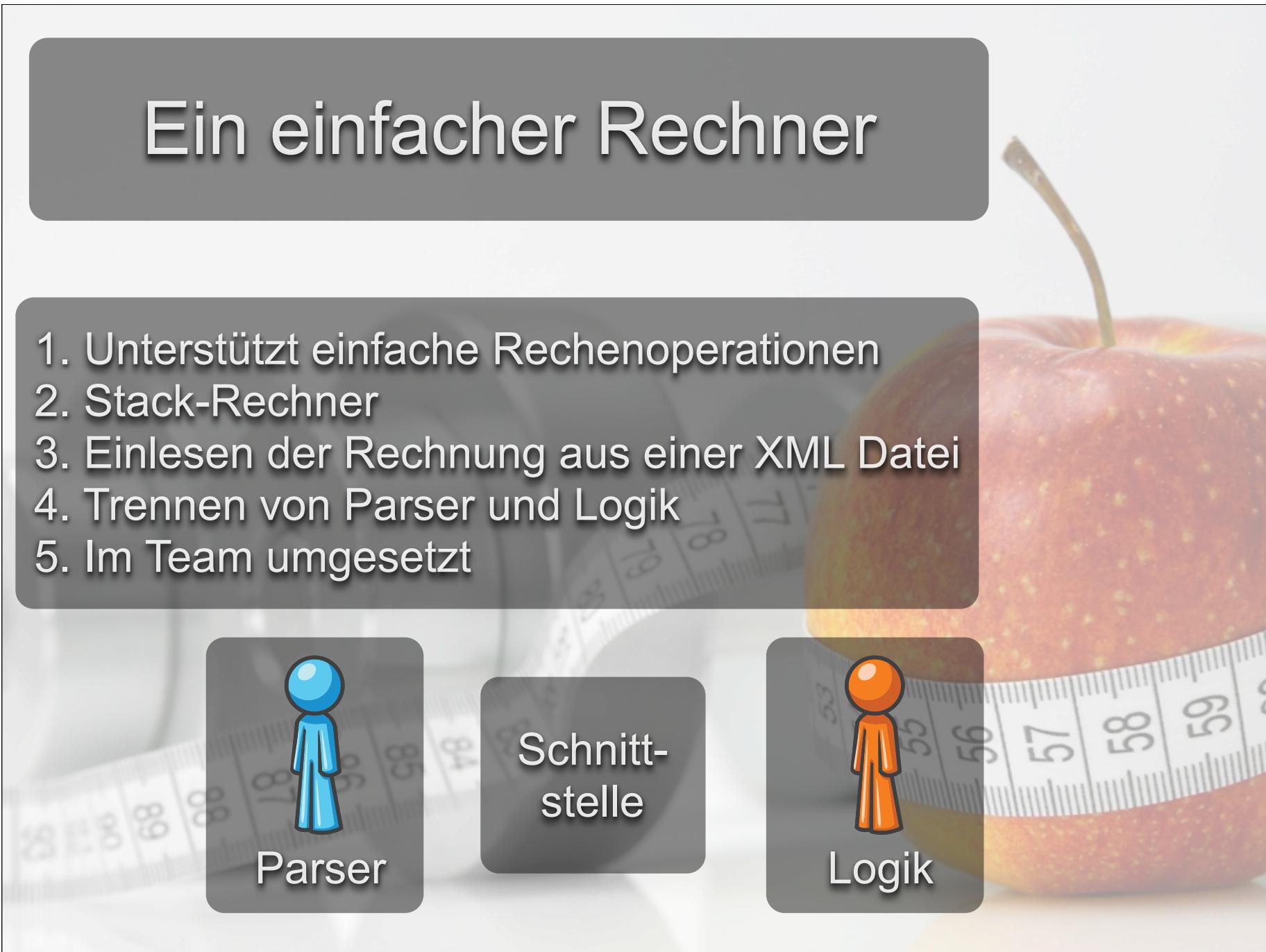


Parser

Schnitt-
stelle



Logik



Beispiel und JUnit Tests...



<https://github.com/gfliess/TrainingJava-Modul3.git>

Refactoring

Der Mut etwas funktionierendes
zu verändern...



*Any fool can write code that
a computer can understand.
Good programmers write
code that humans can
understand.*

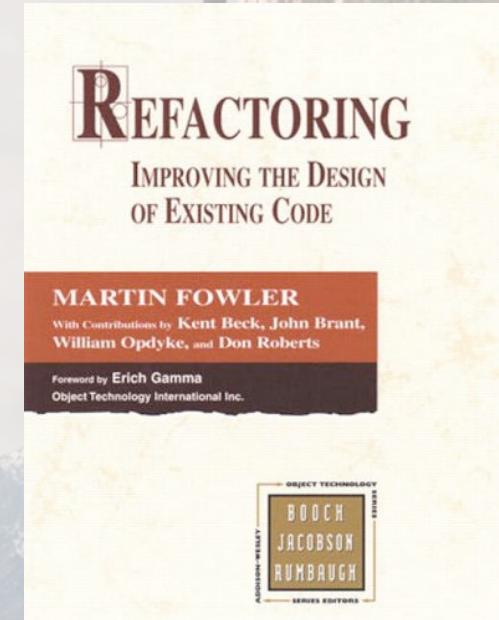
(Fowler)



Martin Fowler definiert an die 70 Refactorings in 7 Gruppen

Ähnlich wie Patterns aufgebaut

- Name
- Motivation
- Umsetzung
- Beispiel



Refactoring

Verbessert

- Struktur
- Architektur
- Lesbarkeit

Ohne die Funktionalität
zu ändern!

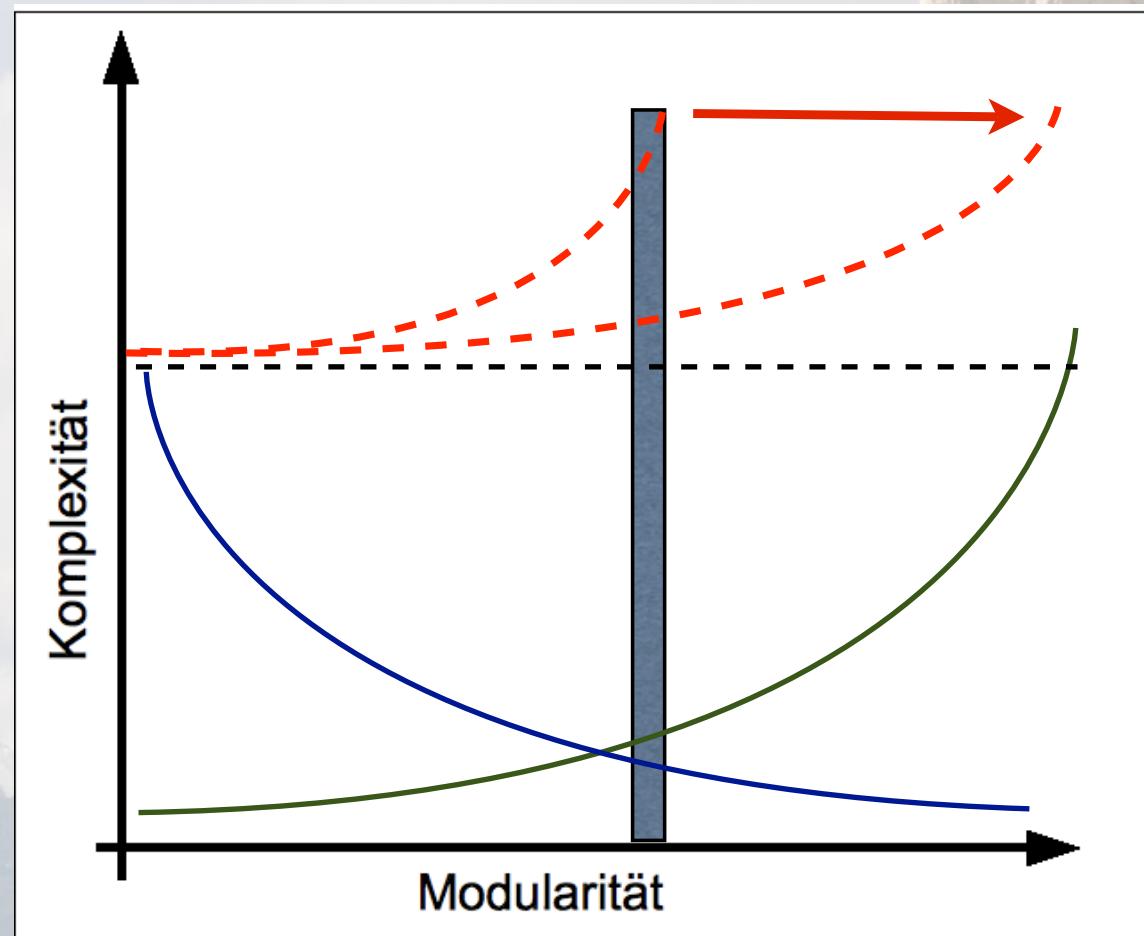


Modularität

summierte
Komplexität

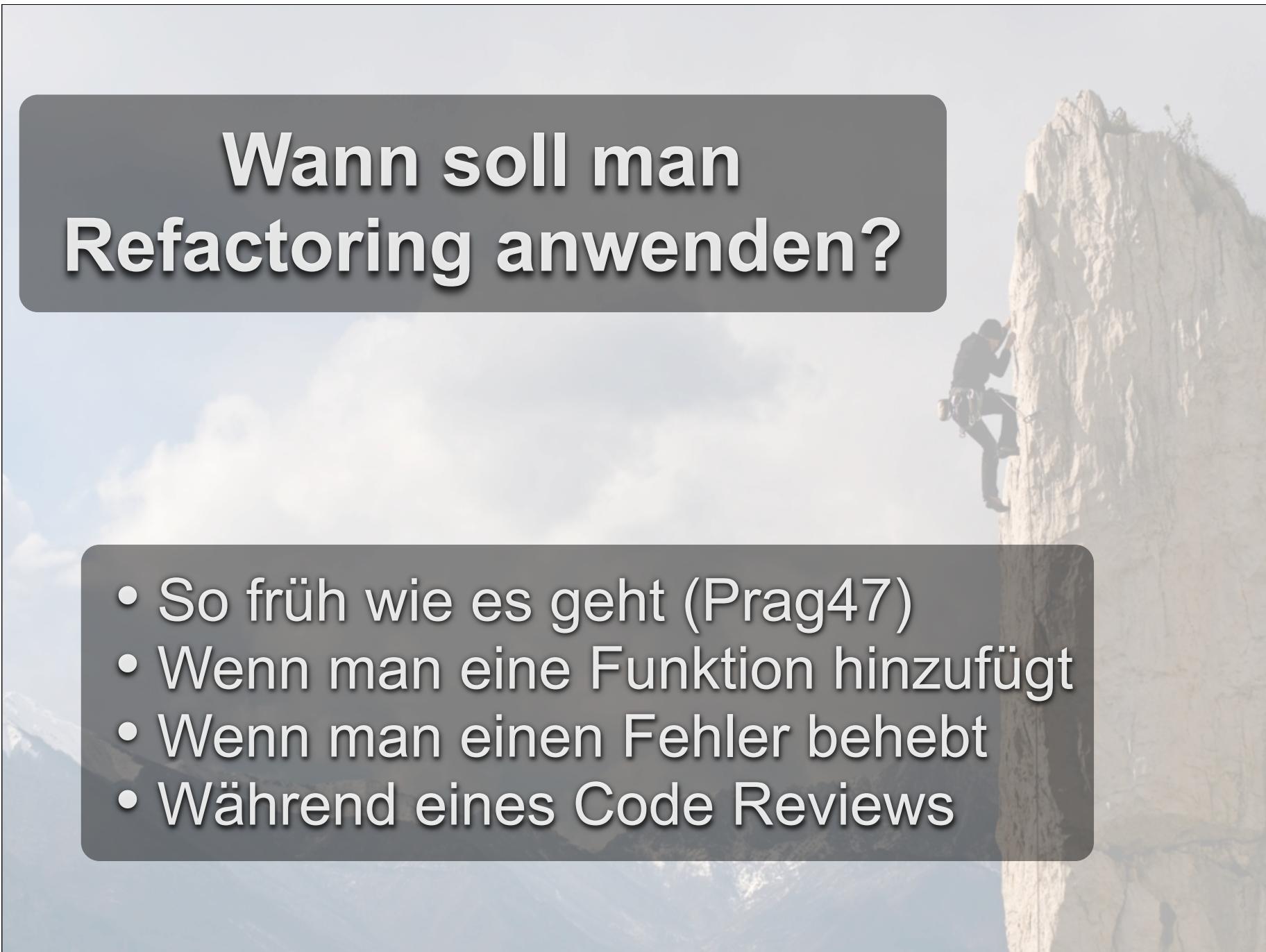
strukturelle
Komplexität

algorithmische
Komplexität



Wann soll man Refactoring anwenden?

- So früh wie es geht (Prag47)
- Wenn man eine Funktion hinzufügt
- Wenn man einen Fehler behebt
- Während eines Code Reviews



Wann soll man Refactoring nicht anwenden?

- Ohne Testfälle
- Kurz vor einer Abnahme
- Wenn es nichts mehr nützt...



Rename Class/ Method/Field

Motivation: Die Namensgebung einer ist verwirrend

- Ändern Sie den Namen
- Bessern Sie das in allen anderen Klassen aus

Encapsulate Field

Motivation: Vermeiden Sie öffentliche
Felder

- Machen Sie die Felder private
- Schreiben Sie Getter und Setter

Extract Class

Motivation: Eine Klasse macht die Arbeit die eigentlich mehrere Klassen machen sollten

- Schreiben Sie eine neue Klasse
- Teilen Sie die Aufgaben

Encapsulate Collection

Motivation: Eine Methode hat eine modifizierbare Collection als Rückgabewert, die nicht modifizierbar sein sollte.

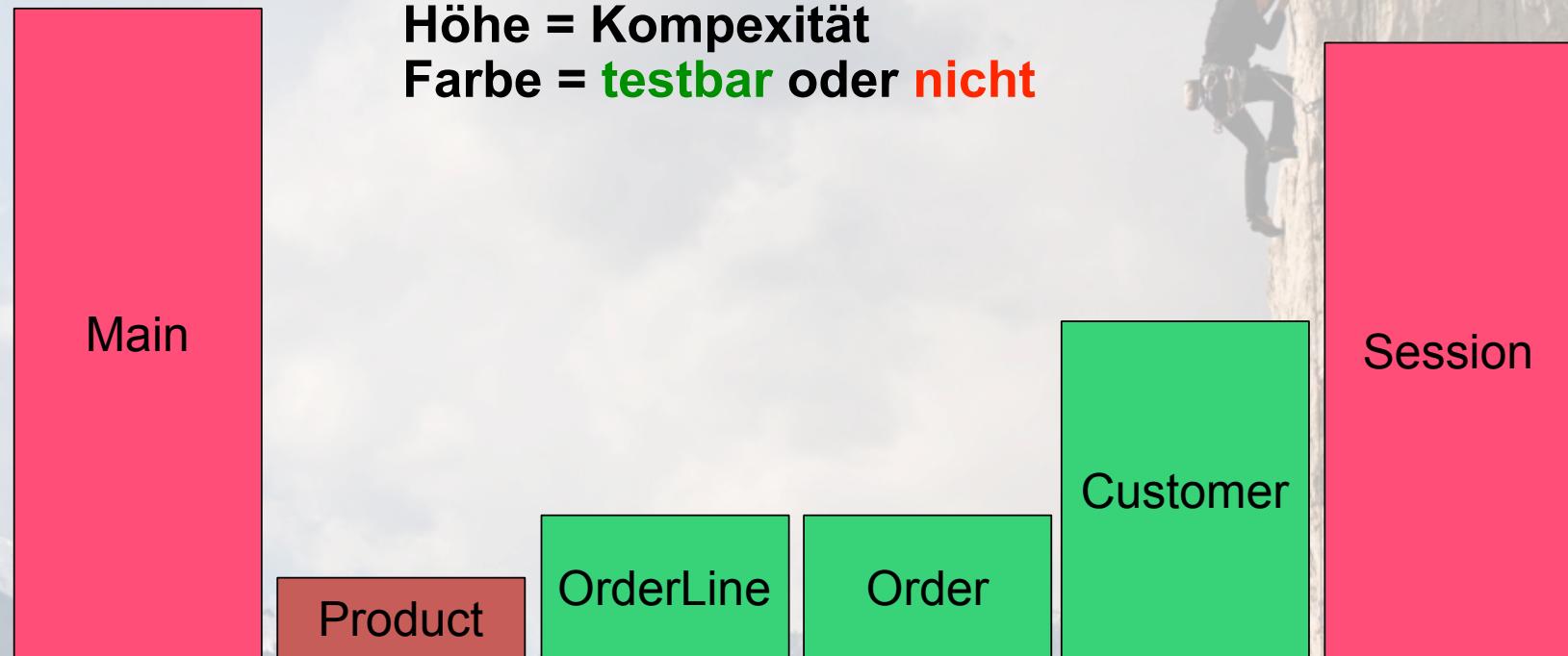
- Machen Sie die Rückgabe nicht modifizierbar
- Fügen sie eine add(<Type>) Methode ein.

Beispiel: YASCLS

- Yet another simple command line shop ;-)
- Quellcode ist im GIT
 - <https://github.com/gfliess/TrainingJava-Modul3-Refactoring.git>
- Folgende Objekte
 - Produkte
 - Bestellzeilen
 - Bestellung
 - Käufer
 - Bestell-Session

Module

Höhe = Kompexität
Farbe = **testbar** oder **nicht**



Remove Setting Method

Motivation: Ein Feld sollte nur bei der Erstellung eines Objektes gesetzt werden können

- Löschen Sie den Setter
- Fügen Sie einen Konstruktor für das Feld ein

Testbarkeit: Testen von Constructor und Gettern

(Product)

Extract Method

Motivation: Aufteilen langer Methoden und Gruppieren von Code der ein Problem löst

- Wenn Sie einen Kommentar benötigen, machen Sie eine Methode!
- DRY (*Prag11:don't repeat yourself*)

Testbarkeit:

- Die algorithmische Komplexität einer Methode wird aufgeteilt.
- Möglichkeit für TestFälle

(Session: calcPrice, createPlain, createXml)

Move Method

Motivation: Eine Methode sollte in einer anderen Klasse angesiedelt sein

- Aufteilen der Zuständigkeiten

Testbarkeit:

- Test muss in neuer Klasse gemacht werden
- Ein Test nur mit einem Thema
- Weniger Klassen sind leichter zu testen.

(Session.calc -> Order)

Hide Delegate

Motivation: Beim Aufruf müssen Sie mit einem Feld des Objektes reden und mit einem Feld und ...

- LoD
- Manchmal fallen Klassen dadurch weg

Testbarkeit: Entkoppelte Module sind leichter zu testen.

(Session -> Customer)

Encapsulate Collection

Motivation: Eine Methode hat eine modifizierbare Collection als Rückgabewert, die nicht modifizierbar sein sollte.

- Machen Sie die Rückgabe nicht modifizierbar
- Fügen sie eine add(<Type>) Methode ein.

Testbarkeit:

- Weiterer Testfall nötig für den Zugriff nötig
- Testfall ob nur gleiche Objekte in der Collection sind fällt weg. (Order.addLine(..)) (tests starten !)

Extract Class

Motivation: Eine Klasse macht die Arbeit die eigentlich mehrere Klassen machen sollten

- Aufteilen der Zuständigkeiten
- Meist in Kombination mit *Move Method*

Testbarkeit: Ermöglicht es manchmal überhaupt erst zu Testen

(Main -> ProductFactory und Parser(inputStream))

Extract Interface

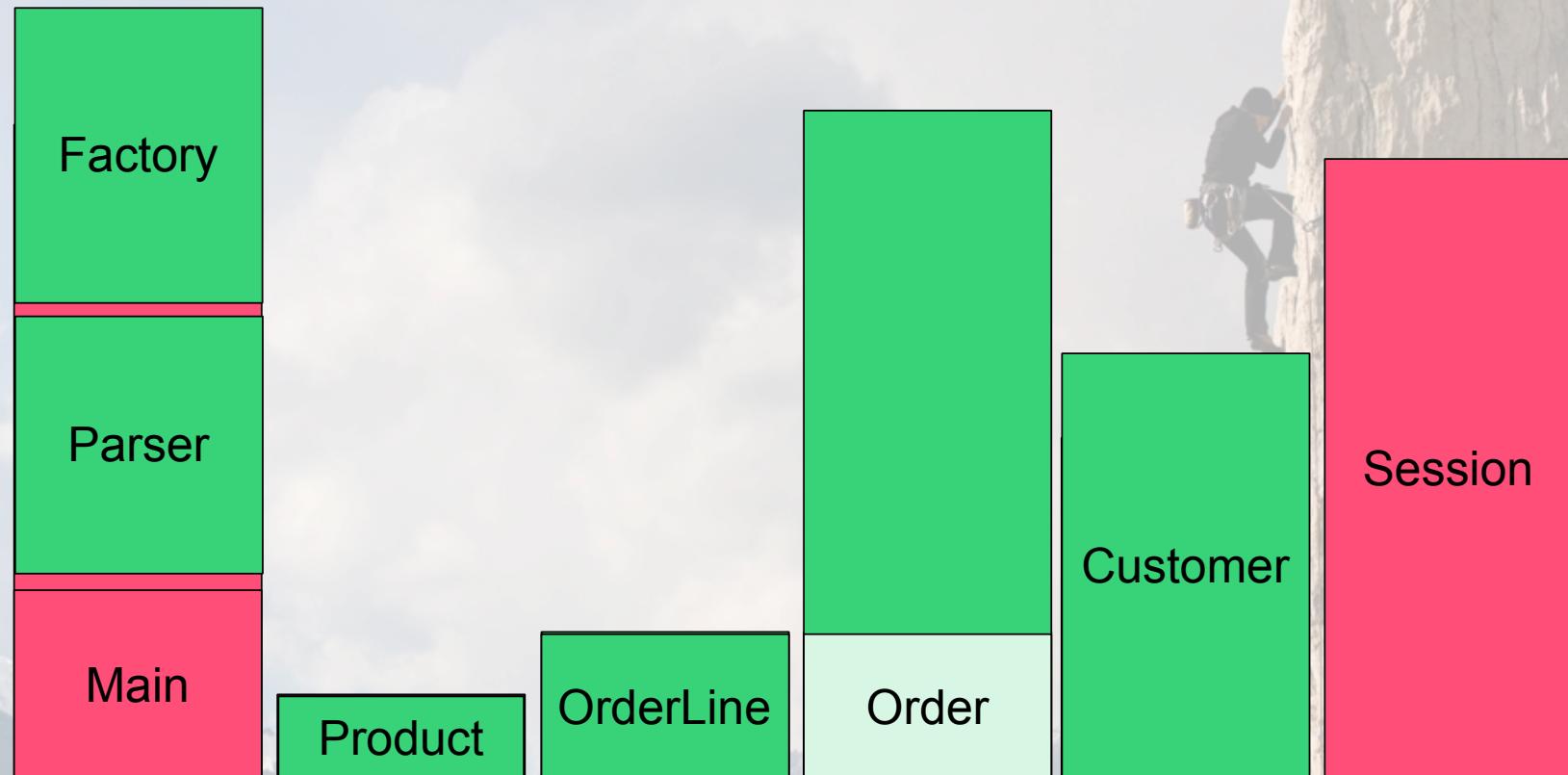
Motivation: Es soll in Zukunft möglich sein, die Implementierung zu tauschen

Testbarkeit: Einführen eines abstrakten Testfalles der das Interface testet

- Nur für Methoden die überall gleich sein sollen
- Allgemein aber nicht immer möglich

(Factory) (zuerst rename und dann extract)

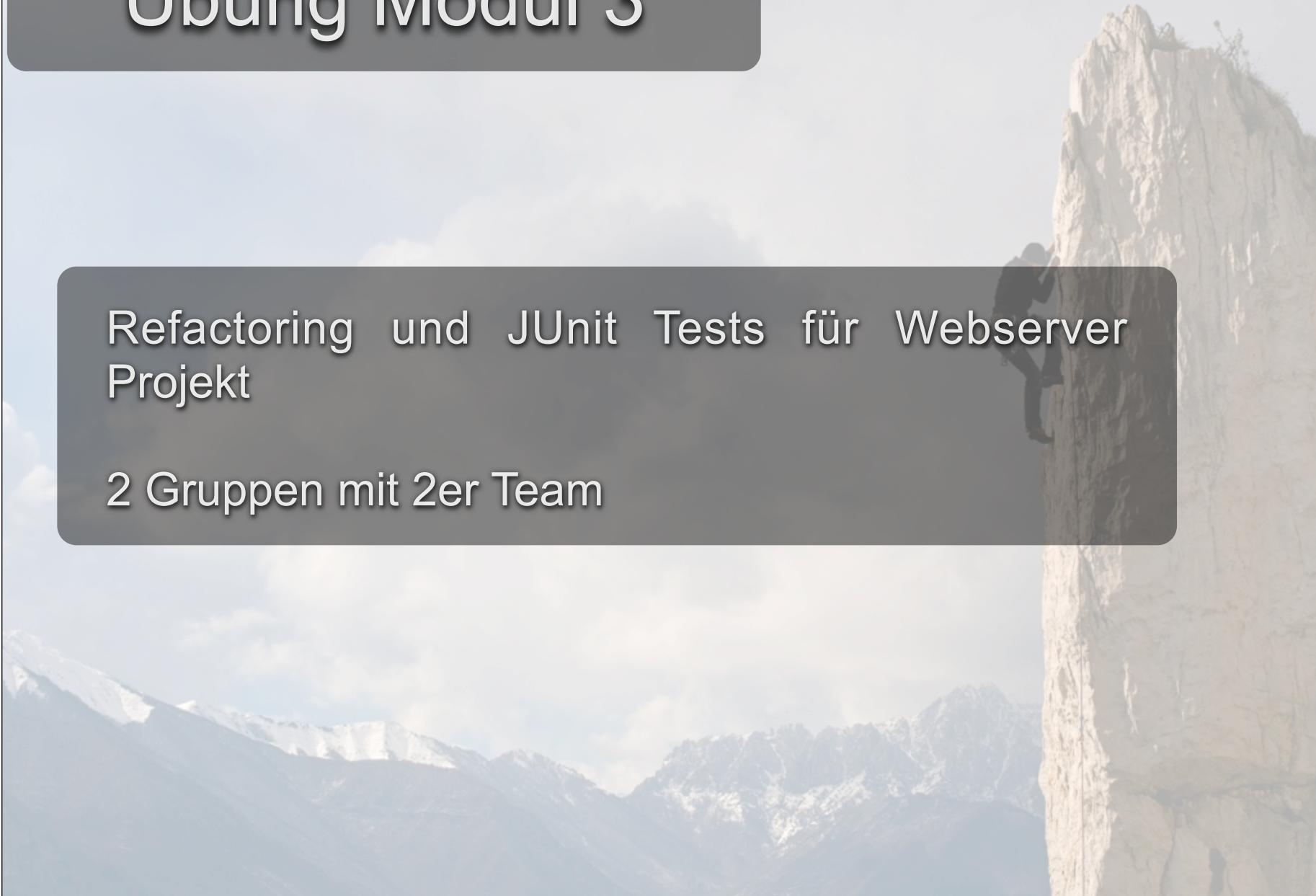
Module neu

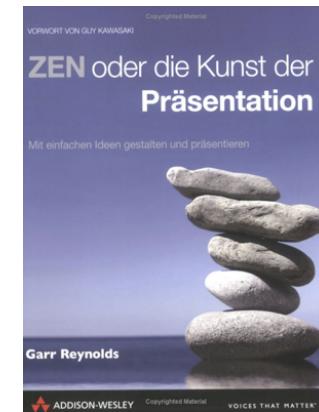
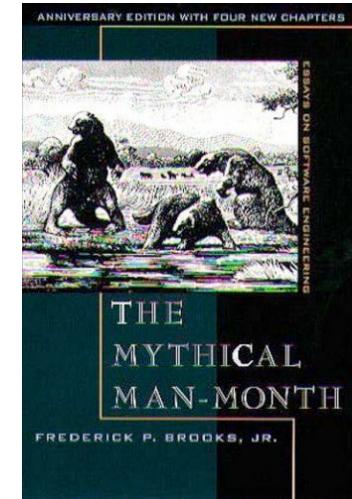
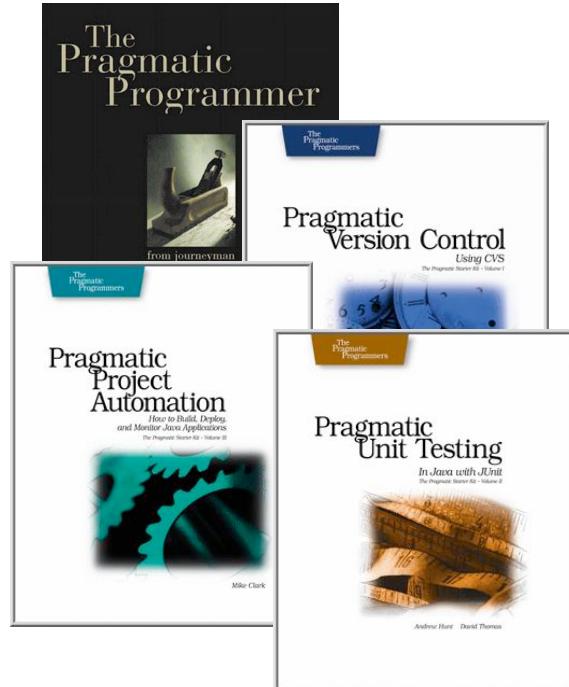
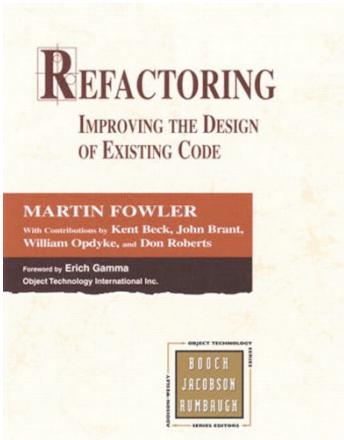


Übung Modul 3

Refactoring und JUnit Tests für Webserver Projekt

2 Gruppen mit 2er Team





<http://red solo.blogspot.com/2008/04/guide-to-building-net-projects-using.html>