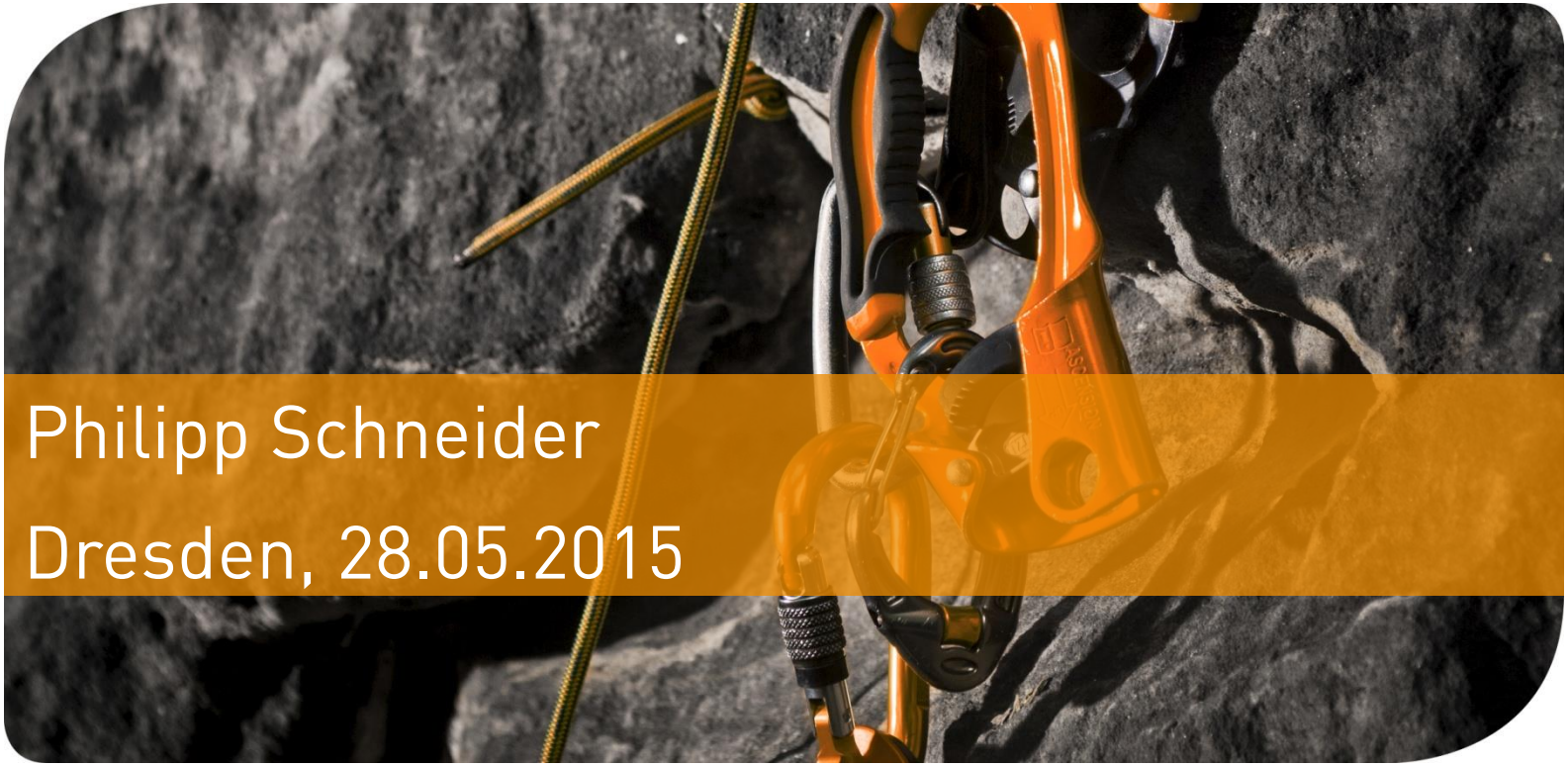
JUG: Agile Scrum – Entwicklungspraktiken

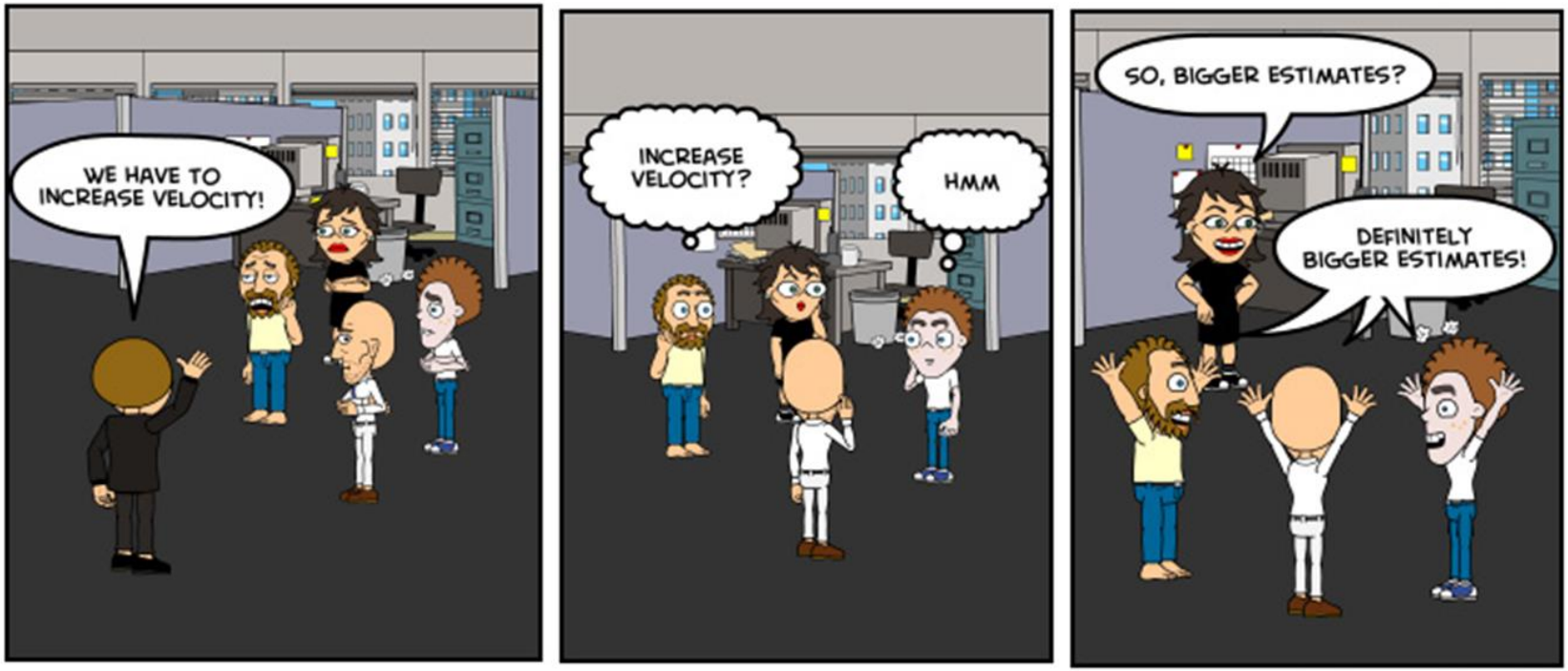
tracetronic
GUIDING YOUR VISION



Philipp Schneider

Dresden, 28.05.2015

2 Scrum Dilemma



Quelle: http://1.bp.blogspot.com/-uztI91DqrNI/U5ijO0mowgI/AAAAAAAAGu0/yrHHA12TYh4/s1600/download.jpg

3 Scrum Entwicklungspraktiken - Mainstream?

Quelle: https://www.verwaltung.fh-koeln.de/imperia/md/images/dez5/sg52/termintipps/2012/raupe_schmetterling.jpg

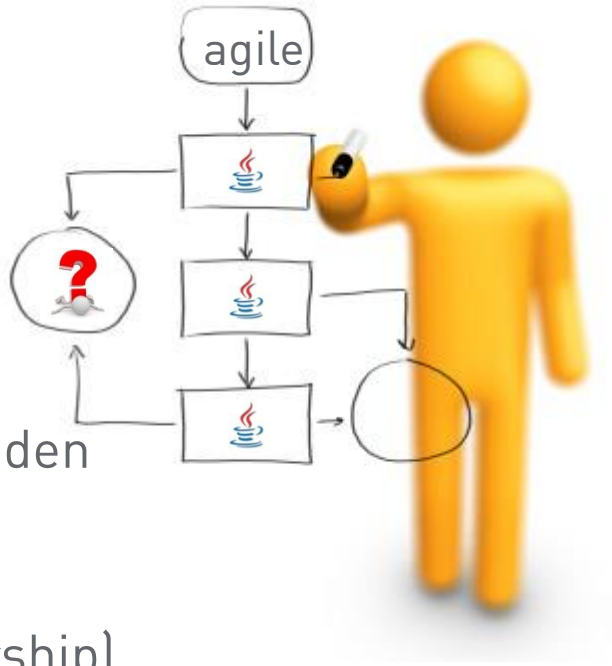
Gliederung

- ❄ Aller Anfang: eine Architekturvision
- ☑ Ohne geht's nicht mehr!

  Continuous Integration
- ♔ Die Königsdisziplin:

  Test-Driven-Development (TDD)
- ♥ Refactoring – Software-Anti-Aging für den Code ab 30 Zeilen
- ✈ Pair-Programming (+Collective Ownership)

  eine Ausbildung wie bei Piloten
- ♈ Katas und Dojos - Entwicklerkampfkunst

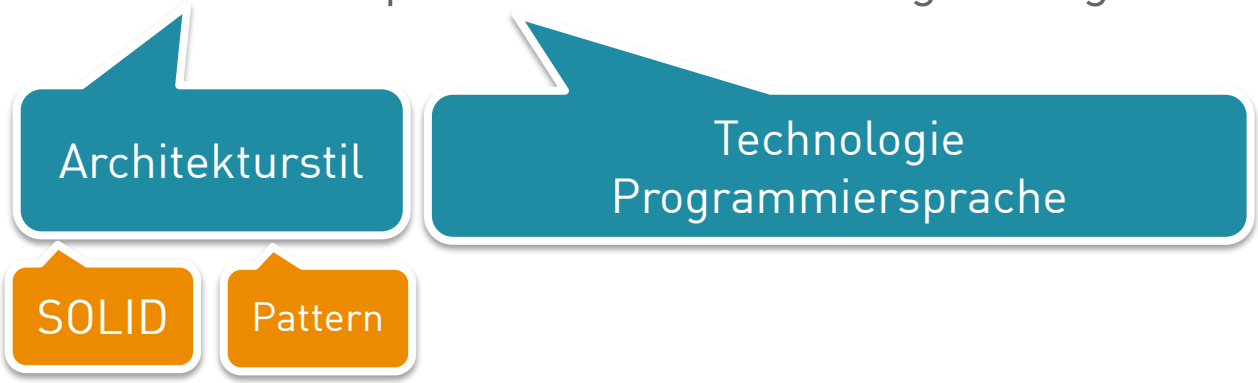


Quelle: http://previews.123rf.com/images/ayzek/ayzek1106/ayzek11060016 9/9805192-Stick-Figure-Drawing-Empty-Organization-Chart-Stock-Photo.jpg

Aller Anfang: eine Architekturvision

Team-Architekturvision:

- leichtgewichtig
- kritische Aspekte für Produkterfolg festlegen



tracetronic
GUIDING YOUR VISION

Eigenschaften einer guten Vision:

Clear – relevante Entscheidungen klar für alle Teammitglieder dokumentiert

Accepted – jedes Teammitglied trägt die Vision

Short – Darstellung kritischer Entscheidungen, KEIN vollständiges Modell

Architektur entwickelt in Sprints:

- nur mit inkrementellen Entwurf möglich
  - vertikale Durchstiche statt Schicht für Schicht
  - Qualitätskriterium: Der existierende Code ist zu jeder Zeit leicht änderbar.

Quelle: https://s-media-cache-ak0.pinimg.com/236x/59/a0/10/59a01093defa6feb404b9003314625cb.jpg

Architekturvision – Vier Fäuste für ein...

Bad Practice

- Detaillierte Architekturvision macht es schwer, iterativ Sprint erworbenes Wissen in die Architektur einfließen zu lassen
- Keine Architektur macht es unmöglich, zielorientiert das erste Produktinkrement zu liefern und birgt die Gefahr hoher Refactoring Aufwände

Quelle: http://coolsandfools.com/wp-content/uploads/2014/04/21-Civil-Engineer-Transportation-Design-Bridge-Fail-500x350.jpg

Zwei Faustregeln

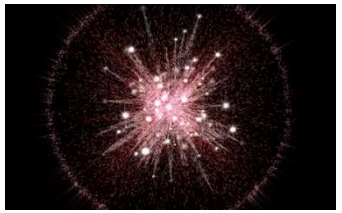
Quelle: http://www.taltextil-shop.de/images/product_images/info_images/669_1.jpg

- Im Zweifel auf architekturelle Festlegungen in der Vision verzichten
- Architektur- und Technologieentscheidungen zeitnah durch Produktinkremente validieren

tracetronic
GUIDING YOUR VISION

Motivation – Big Bang Deployment / Integration Hell

- eigentliche Integration findet ohne CI beim Release erst auf dem Testsystem statt
- Feedback-Schleife nicht gegeben



Quelle: http://scienceline.org/wp-content/uploads/2008/02/bigbang1.jpg

Konsistenzkriterienprüfung via CI

- Compilerfähigkeit
- Semantische Korrektheit
  - Klassentests -› Unit-Tests
  - Komponententests -› Integrationstests
  - „Benutzertests“ -› Akzeptanztests
- Funktionierende Paketierung (jederzeit ein Release)
- Optional – formale Korrektheit
  - Testüberdeckung
  - Codemetriken und Komplexitätsmaße

Buildbot

Jenkins

Hudson

alles automatisiert beim Check-in

tracetronic
GUIDING YOUR VISION

Broken-Windows-Theory – schleichender Verfall

Quelle: http://blog.smartbear.com/wp-content/uploads/2013/03/fix_the_broken_window.jpg

1. CI-Server meldet Fehler
2. Keine Korrektur -› weiterer Fehler wird eingecheckt
3. Kein Zustandswechsel mehr erkennbar
4. Längerer Zeitraum keine Verbesserung
5. Vertrauen in Codebasis sowie Motivation für Systempflege geht verloren
6. Qualitativer Abwärtsstrudel beginnt -› bis zur Unwartbarkeit

tracetronic
GUIDING YOUR VISION

Stop-the-Line Prinzip (Zero-Tolerance-Policy)

Quelle: http://www.nervepoint.com/blog/wp-content/uploads/2015/04/stop.jpg

- Keinen Mängel durch nachgelagerten Bugfix beheben
- Keine weitere Feature-Integration auf dem CI-Server bis alles wieder grün ist
- Team hat ein Verantwortungsbewusstsein für den Build-Zustand

tracetronic
GUIDING YOUR VISION

Quelle: http://www.bestlegalpractices.com/wp-content/uploads/2014/01/10-minute-clock-300x300.jpg

10-Minuten-Builds (XP-Praktik)

- Schnelles Feedback (Build+Tests) -› schnelle Problembehebung
- 10-Minuten Grenze – Null-Toleranz-Richtlinie – sonst tritt *Broken Windows Theory* in Kraft
- Big-System-Problem:
  - Subtests
  - Modularisierung des Builds
  - schnelle CI-Rechner

tracetronic
GUIDING YOUR VISION

Quelle: http://www.secretgardenmontessori.org/picts/branch.gif

Feature-Flags statt Feature-Branches

- Feature-Branches führen zur „Integration Hell“
- *„Feature-Branching is a poor man‘s modular architecture“*

- entkoppelte, modulare Architektur
- Plug-in-Architektur ermöglicht zentrale Feature-Entwicklung und Aktivierung via Feature-Flag

Quelle: http://www.togglz.org/images/togglz-logo.png

Continuous Integration – Scrum Benefit

Scrum-Werte: Offenheit und Mut werden gefördert

Gutes Kriterium für die Defintion of Done

Quelle: http://blog.3back.com/wp-content/uploads/2010/03/TeamPuzzles-Round-Table.jpg

Bereicherung der Sprint-Meetings

Feedback für alle Scrum-Teammitglieder

TDD – Die Köngisdisziplin

Testen ist wirklich wichtig. Wissen wir!

Testen im Programmieralltag. Schnell nervig! Wird bei Stress vernachlässigt, da meist nachgelagert ...

Effektives Testen muss:

- möglichst zeitnah zur Programmierung erfolgen
- automatisiert wiederholbar sein
- Spaß machen!
- so häufig wie das Kompilieren ausgeführt werden
- so einfach wie das Kompilieren sein
- pragmatisch sein
- mehr bringen als kosten

TDD – nach Kent Beck... und die Direktiven

1. Entwickler schreiben automatisierte Tests, während sie programmieren
2. Die Tests werden **VOR** dem zugehörigen Applikationscode geschrieben
3. Design findet in kleinen Portionen und Schritt für Schritt statt

Erste Direktive der testgetriebenen Entwicklung

Motiviere jede Änderung des Programmverhaltens durch einen automatisierten Test!

2. Direktive der testgetriebenen Entwicklung

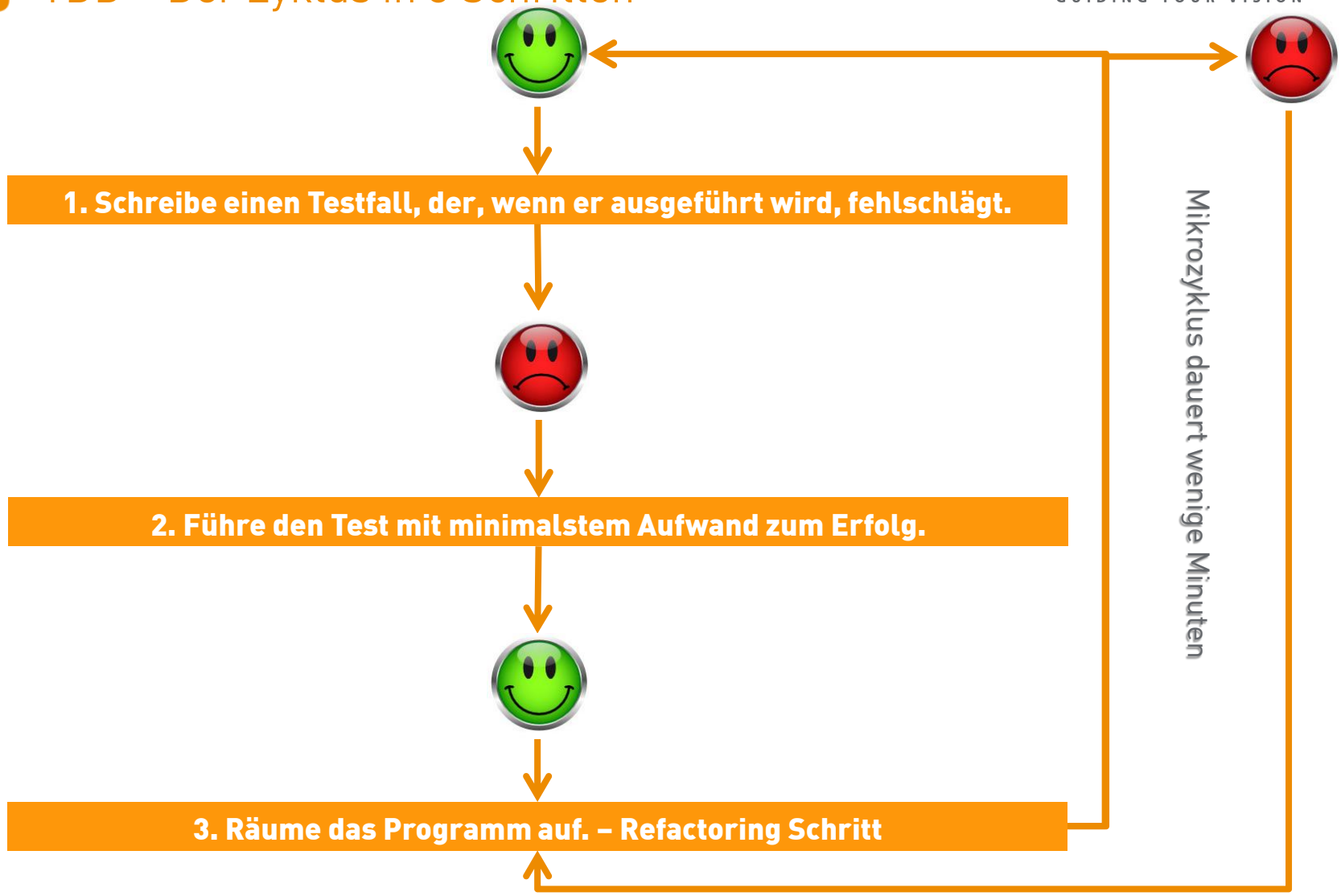
Bringt euren Code immer in die einfache Form!

3. Direktive der testgetriebenen Entwicklung

Integriert euren Code so häufig wie nötig!

TDD – Der Zyklus in 3 Schritten

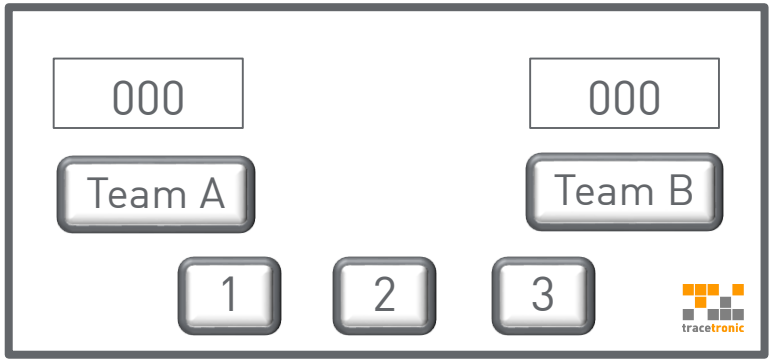
tracetronic
GUIDING YOUR VISION



TDD – Der Basketball-Schiedsrichter

tracetronic
GUIDING YOUR VISION

Produkt: Schiedsrichterzählwerk



- aktuellen Spielstand anzeigen
- einfache Aktualisierung des Spielstandes
- keine Korrekturmöglichkeit

Story: Basketball -ScoreKeeper

- am Anfang steht es 0:0
- Team A macht 1 Punkt
- Team B macht 1 Punkt

- Teams machen jeweils 2 Punkte
- Teams machen jeweils 3 Punkte
- ...

Quelle:
http://www.pxleyes.com/images/contests/smileys/fullsize/smileys_4b9970cd0feb5_hires.jpg

TDD – Der Basketball-Schiedsrichter

tracetronic
GUIDING YOUR VISION

JUnit.org

1. Schreibe einen Testfall, der, wenn er ausgeführt wird, fehlschlägt.

```
public class ScorekeeperTest {
        @Test
        public void initialScoreIs0To0(){
        assertEquals(0,scorekeeper.getTeamAScore());
        }
}
```

- beginne immer bei grünen Tests
- erste Designentscheidung, Name der Klasse, durch den Test getroffen
- aussagekräftiger Name für die Testmethode
- zweite Designentscheidung, Überlegungen zur Schnittstelle, getroffen
- Assert-First Ansatz verwendet

TDD – Der Basketball-Schiedsrichter

tracetronic
GUIDING YOUR VISION

JUnit.org

Story: Basketball -ScoreKeeper
- am Anfang steht es 0:0
- Team A macht 1 Punkt
- Team B macht 1 Punkt

- Teams machen jeweils 2 Punkte
- Teams machen jeweils 3 Punkte
- ...

1. Schreibe einen Tes … lägt.

```
public class S
        @Test
        public
        assertE                                    amAScore());
        }
}
```

- beginne immer bei grünen Tests
- erste Designentscheidung, Name der Klasse, durch den Test getroffen
- aussagekräftiger Name für die Testmethode
- zweite Designentscheidung, Überlegungen zur Schnittstelle, getroffen
- Assert-First Ansatz verwendet

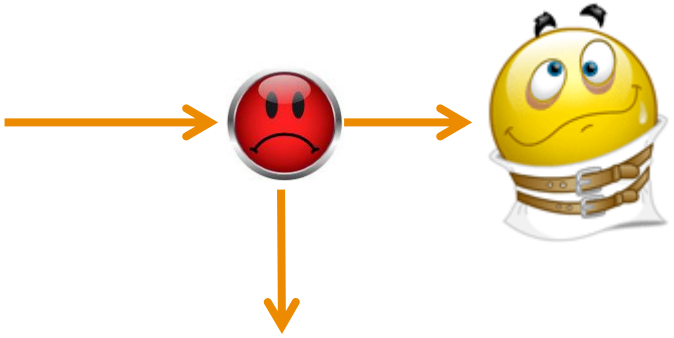
TDD – Der Basketball-Schiedsrichter

tracetronic
GUIDING YOUR VISION

1. Schreibe einen Testfall, der, wenn er ausgeführt wird, fehlschlägt.

```
public class ScorekeeperTest {
        @Test
        public void initialScoreIs0To0(){
        Scorekeeper scorekeeper = new Scorekeeper();
        assertEquals(0,scorekeeper.getTeamAScore());
        }
}
```

```
public class Scorekeeper {
        public Object getTeamAScore() {
        // TODO Auto-generated method stub
        return null;
        }
}
```



Wir benötigen eine **Paranoia-Haltung**!

TDD – Der Basketball-Schiedsrichter

tracetronic
GUIDING YOUR VISION

2. Führe den Test mit minimalstem Aufwand zum Erfolg.

```
public class Scorekeeper {
        // assertEquals(0,scorekeeper.getTeamAScore());
        public Object getTeamAScore() {
                return 0;
        }
}
```

Warum nicht gleich den „richtigen“ Code schreiben?

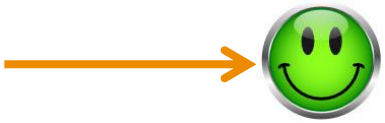
1. wir wollen schnell an einen Punkt kommen, wo alle Tests grün sind
2. für alle anstehenden Aufräum-/Verbesserungsarbeiten (Refactorings) besitzen wir nun ein Sicherheitsnetz
3. Entwurfsentscheidungen hinauszögern, um durch weitere Tests eine Chance das „richtige“ Design zu erhalten, zu bekommen

TDD – Der Basketball-Schiedsrichter

tracetronic
GUIDING YOUR VISION

3. Räume das Programm auf. – Refactoring Schritt

```
public class Scorekeeper {
        private int teamAScore = 0;
        public int getTeamAScore() {
                return teamAScore;
        }
}
```



Wir räumen auf – aber wie viel?

- das ist eine Frage des Geschmacks
- je mehr Vertrauen man in die Änderbarkeit des eigenen Codes hat, desto eher wird man konkrete Entscheidungen auf später verschieben

tracetronic
GUIDING YOUR VISION

Wir ergänzen den Test

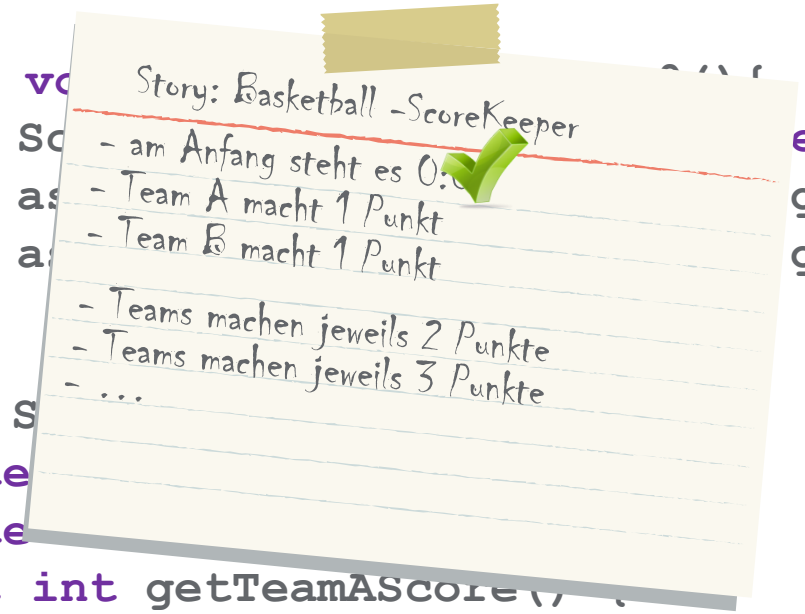
```
public class ScorekeeperTest {
        @Test
        public void initialScoreIs0To0(){
                Scorekeeper scorekeeper = new Scorekeeper();
                assertEquals(0,scorekeeper.getTeamAScore());
                assertEquals(0,scorekeeper.getTeamBScore());
        }
}
public class Scorekeeper {
        private int teamAScore = 0;
        private int teamBScore = 0;
        public int getTeamAScore() {
                return teamAScore;
        }
        public int getTeamBScore() {
                return teamBScore;
        }
}
```

einfache Symmetrie -› neues Domainobjekt vielleicht später

tracetronic
GUIDING YOUR VISION

Wir ergänzen den Test

```
public class ScorekeeperTest {
        @Test
        public vo
                Sc                                    ew Scorekeeper();
                as                                    getTeamAScore());
                as                                    getTeamBScore());
        }
}
public class S
        private
        private
        public int getTeamAScore() {
                return teamAScore;
        }
        public int getTeamBScore() {
                return teamBScore;
        }
}
```



einfache Symmetrie -> neues Domainobjekt vielleicht später

tracetronic
GUIDING YOUR VISION

Nächster Testfall: Team A macht 1 Punkt

```
public class ScorekeeperTest… {
        @Test
        public void teamAScoresOnePoint(){
                Scorekeeper scorekeeper = new Scorekeeper();
                scorekepper.teamAClicked();
                scorekeeper.score1Clicked();
                assertEquals(1,scorekeeper.getTeamAScore());
                assertEquals(0,scorekeeper.getTeamBScore());
        }
}
```

Designentscheidung: GUI-Entkopplung vom Modell – Sicht des Benutzers eingenommen

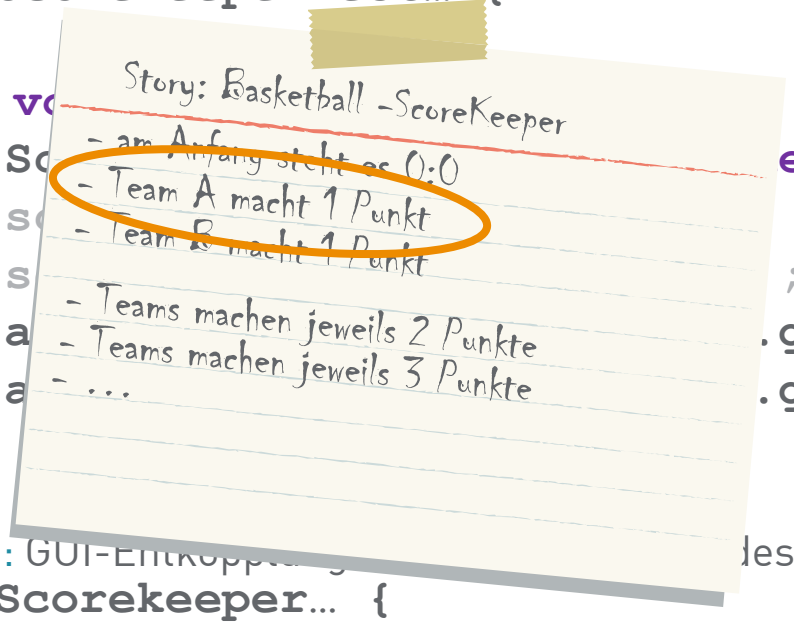
```
public class Scorekeeper… {
        private int teamAScore = 0;
        public void teamAClicked(){};
        public void score1Clicked(){
                teamAScore++;
        }
}
```

alle Tests wieder grün

tracetronic
GUIDING YOUR VISION

Nächster Testfall: Team A macht 1 Punkt

```
public class ScorekeeperTest… {
        @Test
        public vo
                Sc                                    ew Scorekeeper();
                s
                s                                     ;
                a                                     .getTeamAScore());
                a                                     .getTeamBScore());
        }
}
```



Designentscheidung: GUI-Entkopp… …des Benutzers eingenommen

```
public class Scorekeeper… {
        private int teamAScore = 0;
        public void teamAClicked(){};
        public void score1Clicked(){
                teamAScore++;
        }
}
```

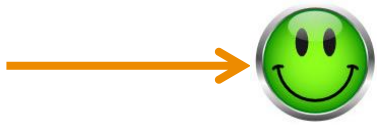
alle Tests wieder grün

TDD – Der Basketball-Schiedsrichter

tracetronic
GUIDING YOUR VISION

3. Schritt: Aufräumen - „Team A macht 1 Punkt“ – auch den Testcode!

```
public class ScorekeeperTest {
        private Scorekeeper scorekeeper;
        @Before public void init(){scorekeeper = new Scorekeeper();};
        @Test public void initialScoreIs0To0(){
                assertScore(0,0);
        }
        @Test public void teamAScoresOnePoint(){
                scorekeeper.teamAClicked();
                scorekeeper.score1Clicked();
                assertScore(1,0);
        }
        private void assertScore(int expAScore,int expBScore){
                assertEquals("score team A", expAScore,
                                scorekeeper.getTeamAScore());
                assertEquals("score team B", expBScore,
                                scorekeeper.getTeamBScore());}}
```



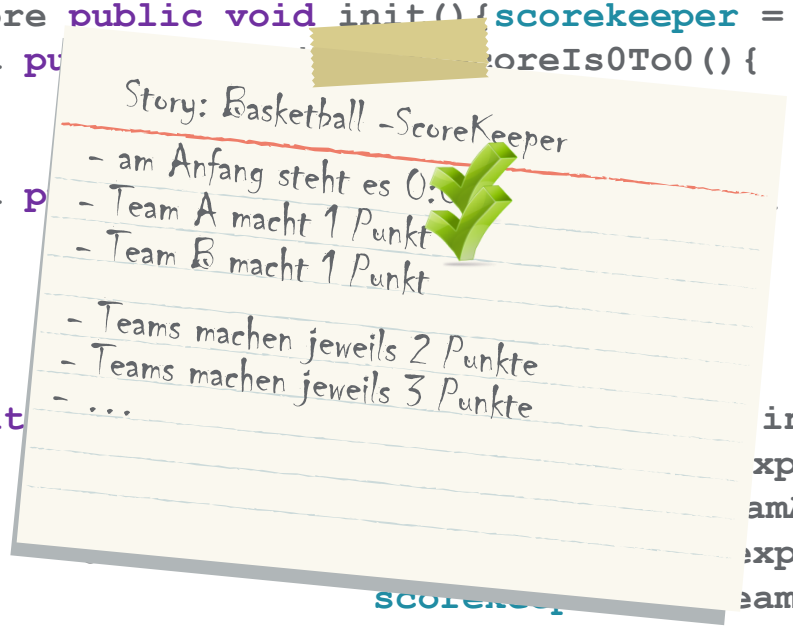
- viele Duplikate im TestCode -› Redundanzen liefern viele fehlgeschlagene Tests auf einmal
- Debugger wird bei aussagekräftigen Fehlermeldungen im Test fast überflüssig

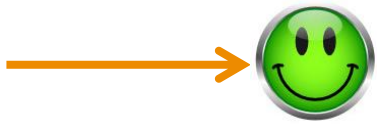
TDD – Der Basketball-Schiedsrichter

tracetronic
GUIDING YOUR VISION

3. Schritt: Aufräumen - „Team A macht 1 Punkt“ – auch den Testcode!

```
public class ScorekeeperTest {
        private Scorekeeper scorekeeper;
        @Before public void init(){scorekeeper = new Scorekeeper();};
        @Test pu                                    oreIs0To0(){

        }
        @Test p

        }
        privat                                                  int expBScore){
                                                                xpAScore,
                                                                amAScore());
                                                                xpBScore,
                                                                eamBScore());}}
```





- viele Duplikate im TestCode -› Redundanzen liefern viele fehlgeschlagene Tests auf einmal
- Debugger wird bei aussagekräftigen Fehlermeldungen im Test fast überflüssig

TDD – aufhören, wenn es am schönsten ist

tracetronic
GUIDING YOUR VISION

TDD ist eine der wichtigsten und mächtigsten Entwicklungspraktiken in der agilen Welt

- Legacy Code entspricht jedem Code ohne Unit-Tests
- Fokussierung (Scrum-Wert) auf das Wesentliche
- Entwurf wird schrittweise vorangetrieben
- Debugger wird überflüssig

„Dieser TDD Zyklus funktioniert vielleicht bei einfachen und unabhängigen Objekten, aber im wirklichen Leben hängen Objekte von anderen Objekten ab und diese wiederrum von anderen. Wie soll ich denn dann die ganzen Delegationen testen? ...“

Lösung: Mock-Frameworks – empfehlenswert für das Selbststudium!

Refactoring: Software-Anti-Aging

tracetronic
GUIDING YOUR VISION

Ein Nerd zum anderen Nerd: *„Und dann habe ich diese Klasse aufgemacht. Alter Schwede, was für ein Schweinkram! Das hättest du mal sehen sollen! 4500 Zeilen war die lang, Dutzende Methoden, eine davon über 350 Zeilen lang. Bei vielen Zeilen muss man horizontal scrollen, um sie lesen zu können. Eine Kaskade von if-else-Verschachtelungen, ganz viel redundanter …“*

Wichtigstes Refactoring Kriterium
Don‘t repeat yourself (DRY) !!!

Warum? Wie? Wann?

- **Warum**: keine Refactorings führen zu technischen Schulden -› Veralterung -› Neuentwicklung
- **Wie**: in kleinen (auch experimentellen) Schritten auf dem grünen Pfad der Hängematte
- **Wann**: „JETZT!“ – mit jeder Minute die vergeht, erhöht sich die Chance den Schlammball wachsen zu lassen -› gutes DoD Kriterium: „vollständig refaktorisiert“

Refactoring: Basis-Refactorings

tracetronic
GUIDING YOUR VISION

Umbenennen - selbsterklärende Namen sind essenziell für das Verständnis

```
int a = 378;                                    int tageBisZurHochzeit = 378;
```

Verschieben - wenn ein Teil der Software fehl am jetzigen Platz ist

Extrahieren - Verantwortlichkeiten klarer trennen

```
public int zähleTTDiplomanten(List<TTPerson> persons){
  int anzahl=0;
  for(TTPerson each : persons){
       if(each.istStudent() && each.hatZweiBildschirme()) anzahl++;}
return anzahl;}
```

Inlinen – Gegenteil von Extrahieren – Bsp. folgt

Löschen – auskommentierten bzw. toten Code entfernen

Refactoring: Basis-Refactorings

tracetronic
GUIDING YOUR VISION

Umbenennen - selbsterklärende Namen sind essenziell für das Verständnis

```
int a = 378;
```

```
int tageBisZurHochzeit = 378;
```

Verschieben - wenn ein Teil der Software fehl am jetzigen Platz ist

Extrahieren - Verantwortlichkeiten klarer trennen

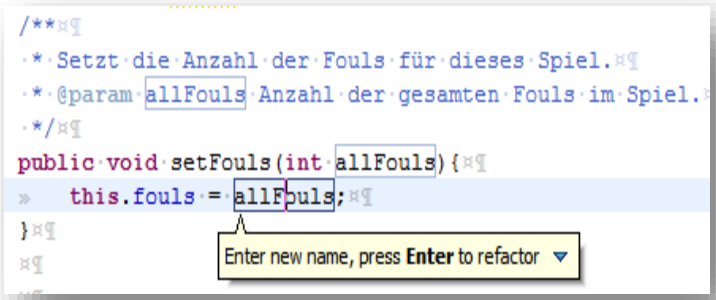
```
public int zähleTTDiplomanten(List<TTPerson> persons){
  int anzahl=0;
  for(TTPerson each : persons){
    if(istDiplomant(each))anzahl++;}
return anzahl;}

private boolean istDiplomant(TTPerson person){
  return person.istStudent() && person.hatZweiBildschirme();}
```

Inlinen – Gegenteil von Extrahieren – Bsp. folgt

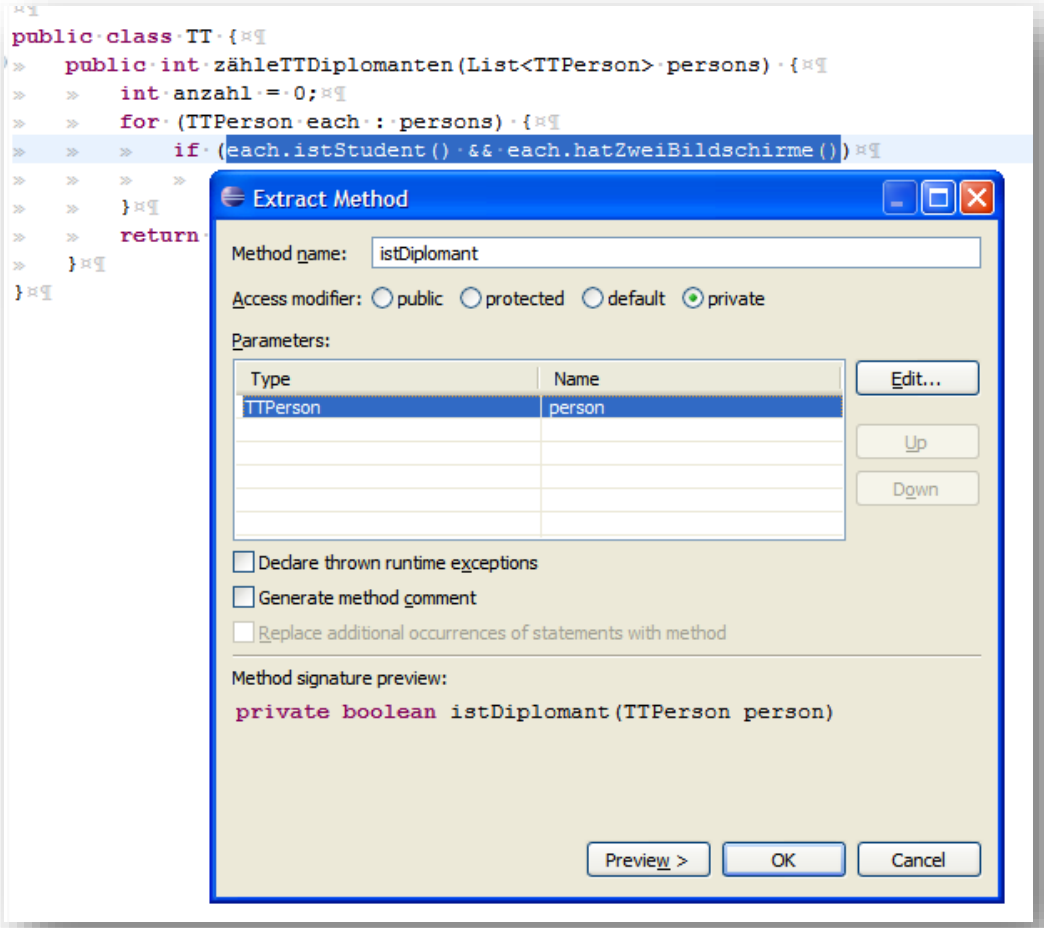
Löschen – auskommentierten bzw. toten Code entfernen

Automatisierte Refactorings

Der Klassiker: Rename



Extrahieren – am Bsp. Extract Method



Automatisierte Refactorings

Geheimwaffe „Inline“

```
public class InlineDrucker{
  private Drucker drucker;
  …
  public void drucke(Dokument doc){
     int anzahlSeiten = doc.getAnzahlSeiten();
     drucker.drucke(doc,1,anzahlSeiten);
  }
  public void druckeVonBis(Dokument doc, int von, int bis){
     drucker.drucke(doc,von,bis);
  }
}
```

Automatisierte Refactorings

Geheimwaffe „Inline“

```
public class InlineDrucker{
  private Drucker drucker;
  …
  public void drucke(Dokument doc){
     this.druckeVonBis(doc,1,doc.getAnzahlSeiten());
  }
  public void druckeVonBis(Dokument doc, int von, int bis){
     drucker.drucke(doc,von,bis);
  }
}
```

Automatisierte Refactorings

Geheimwaffe „Inline“

```
public class InlineDrucker{
  private Drucker drucker;
  …
  public void druckeVonBis(Dokument doc, int von, int bis){
     drucker.drucke(doc,von,bis);
  }
}
```

Alter Aufruf:

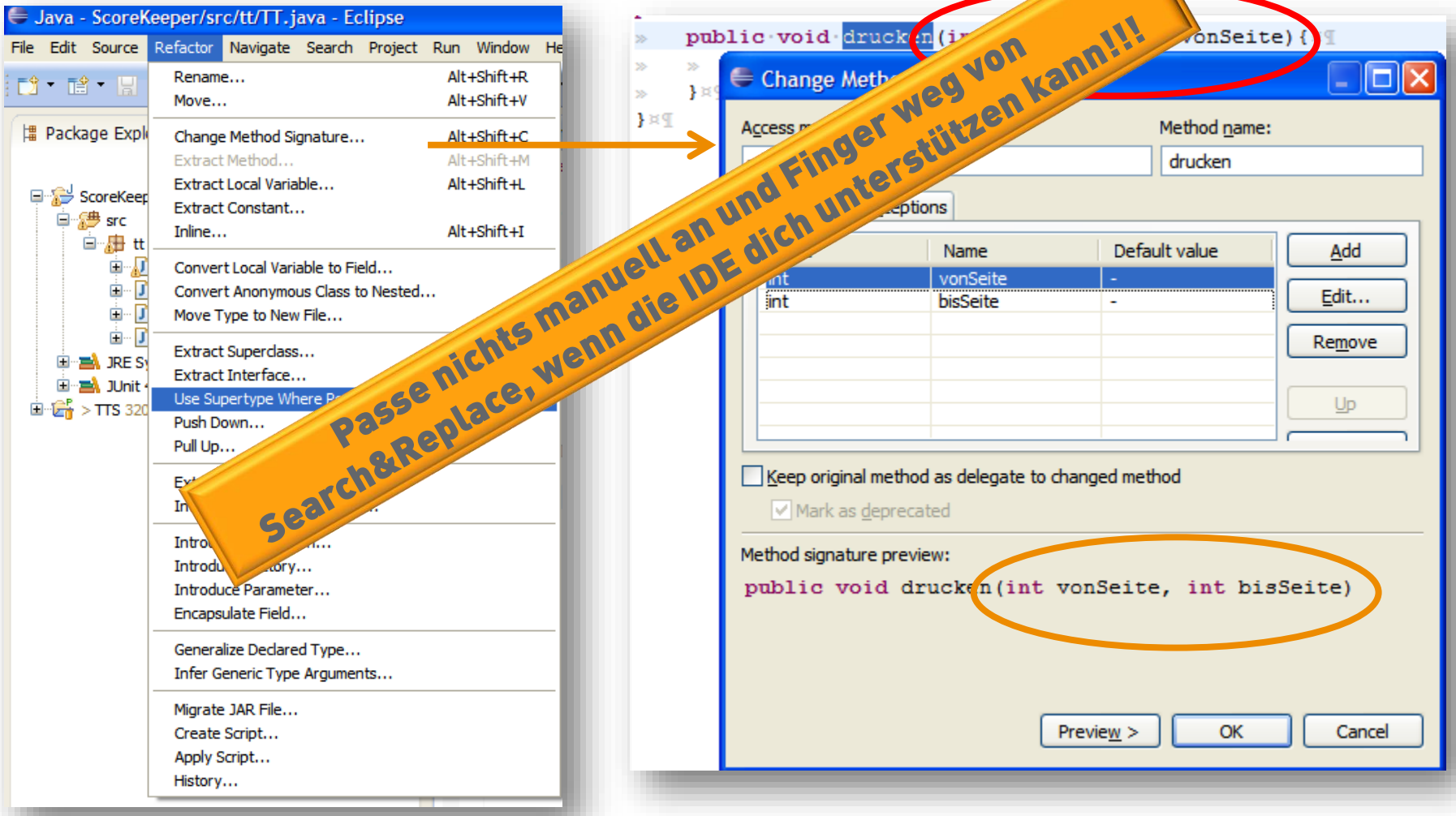
```
example.drucke(dokument);
```

systemweit durch die neue Methode ersetzt:

```
example.druckeVonBis(dokument,1,
dokument.getAnzahlSeiten());
```

Automatisierte Refactorings

IDE-Möglichkeiten



Pair-Programming: nur Fliegen ist schöner

uddha.com/wp-content/uploads/2012/12/PairProgrammingInAction.jpg

„Keine Angst vor Pair-Programming. Du bist nicht so gut, wie du glaubst, aber auch nicht so schlecht, wie du befürchtest!“

- ständiger Dialog über das Detaildesign und den Code
- Rollen wechseln ständig
- Entwickler mit der Tastatur nennt man Pilot, dieser programmiert
- der Andere ist der Navigator und beobachtet, macht Anmerkungen und gibt Hinweise -› fordert auch Tastatur
- aller paar Minuten findet ein Wechsel an der Tastatur und damit der Rollen statt
- Partnerwechsel meist täglich
- Dauer- und Exklusivpaarungen zum Wissensaustausch angebracht
- Freude an der Arbeit

Pair-Programming: Vorteile und Herausforderungen

Quelle: http://www.remoters.de/wp-content/uploads/2013/03/vorteil.jpg

- Freude an der Arbeit
- höhere Disziplin
- besserer Code
- Verteilung des Wissens
- Teambildung
- weniger Unterbrechungen

Quelle: http://www.abzonline.de/news/pics/640x360/mitarbeiter-stossen-ihren-ideen-selten-ablehnung-nachteil-betrieb-foto-fotolia_50426.jpg

- höhere Entwicklerkosten
- Teamfindung kann am Anfang problematisch sein
- Autoritätsprobleme
- zeitliche Belastung bei unterschiedlichen Wissensständen

Collective Ownership

Strict Ownership – konkrete Entwicklerzuweisung

Weak Ownership – jeder darf alles und es gibt Hauptverantwortliche für gewisse Komponenten

No Ownership – Verantwortung für den Quellcode nicht ausreichend klar – faktisch nicht vorhanden

Collective Ownership – Quellcode ist Teamhoheit

Entwickler können jede Quelltextstelle ändern -> anspruchsvoll

enorme Vorteile: weniger Überlastung Einzelner, sowie viel leichtere Planungen im Sprint und während der Daily Scrums

Quelle: http://agilepainrelief.com/images/WindowsLiveWriter/team_work.jpg

gemeinsames Verständnis und gleiche Sprache notwendig **Empfehlung: Domain-Driven Design**

tracetronic
GUIDING YOUR VISION

41 Code-Katas

Code-Katas sind kleine vorgegebene Programmieraufgaben

Code-Katas helfen, die Programmierfähigkeiten zu verbessern

Übung macht den Meister

Beispiele:

1. Fibonacci-Reihe berechnen
2. Primfaktorzerlegung
3. Umwandlung römischer Zahlen in arabische
4. ...

Explorative Katas

- Ziel: verstehen, wie sich bestimmte Festlegungen auf das Vorgehen und das Ergebnis auswirken
- Bsp.: Wie verhält sich der Code bei Primfaktorzerlegung bei rein funktionaler Entwicklung?

Einübende Katas

- Wiederholung desselben Lösungsansatzes bis zur Perfektion
- Beispiele:
  - kleinschrittigeres Vorgehen bei TDD
  - Verwendung automatisierter Refactorings
  - Shortcuts der IDE üben

Einfach loslegen! Feedback in der Firma oder im Internet einholen.

„Philipp! Der Vortrag **MUSS** auch Python Beispiele enthalten!“

tracetronic
GUIDING YOUR VISION

42 Code-Katas: FizzBuzz (Einstiegs-Kata)

1. alle Zahlen von 1 bis 100 ausgeben
2. wenn die Zahl durch 3 teilbar, dann Wort Fizz anstatt Zahl ausgeben
3. wenn die Zahl durch 5 teilbar, dann Wort Buzz anstatt Zahl ausgeben
4. wenn die Zahl sowohl durch 3 als auch durch 5 teilbar ist, dann Wort FizzBuzz ausgeben

```
count = 1
while count < 101:
    if count%5 == 0 and count%3 == 0:
        print "FizzBuzz"
        count = count + 1
    elif count % 3 == 0:
        print "Fizz"
        count = count + 1
    elif count % 5 == 0:
        print "Buzz"
        count = count + 1
    else:
        print count
        count = count + 1
```

```
count = 1
while count < 101:
    if count%5 == 0 and count%3 == 0:
        print "FizzBuzz"
    elif count % 3 == 0:
        print "Fizz"
    elif count % 5 == 0:
        print "Buzz"
    else:
        print count
    count = count + 1
```

„Philipp! Der Vortrag **MUSS** auch Python Beispiele enthalten!“

tracetronic
GUIDING YOUR VISION

43 Code-Katas: FizzBuzz (Einstiegs-Kata)

1. alle Zahlen von 1 bis 100 ausgeben
2. wenn die Zahl durch 3 teilbar, dann Wort Fizz anstatt Zahl ausgeben
3. wenn die Zahl durch 5 teilbar, dann Wort Buzz anstatt Zahl ausgeben
4. wenn die Zahl sowohl durch 3 als auch durch 5 teilbar ist, dann Wort FizzBuzz ausgeben

```
count = 1
while count < 101:
    if count%5 == 0 and count%3 == 0:
        print "FizzBuzz"
        count = count + 1
    elif count % 3 == 0:
        print "Fizz"
        count = count + 1
    elif count % 5 == 0:
        print "Buzz"
        count = count + 1
    else:
        print count
        count = count + 1
```

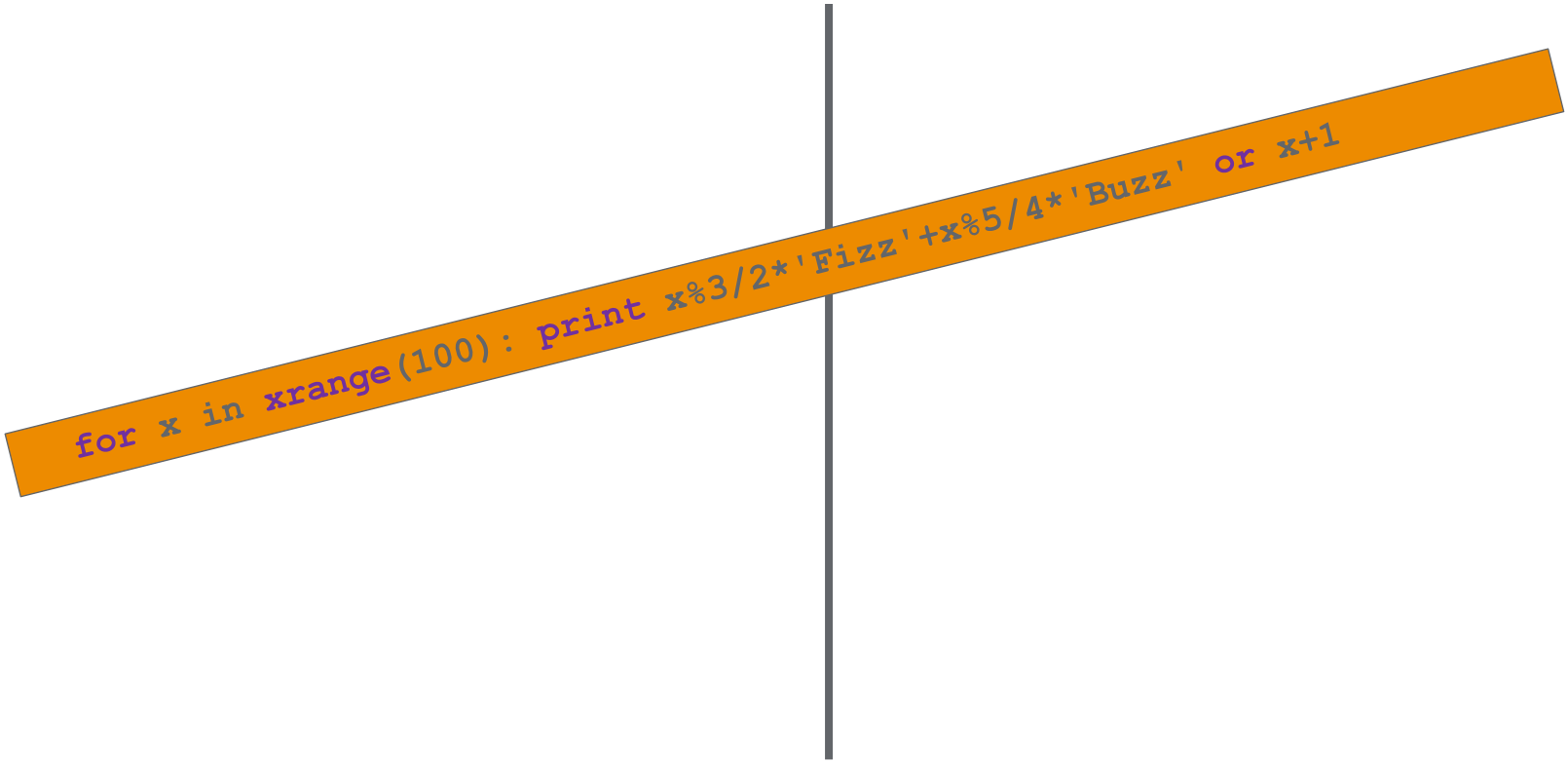
```
for num in xrange(1,101):
    msg = ''
    if num % 3 == 0:
        msg += 'Fizz'
    if num % 5 == 0:
        msg += 'Buzz'
    print msg or num # num Standard
```

„Philipp! Der Vortrag **MUSS** auch Python Beispiele enthalten!“

tracetronic
GUIDING YOUR VISION

Code-Katas: FizzBuzz (Einstiegs-Kata)

1. alle Zahlen von 1 bis 100 ausgeben
2. wenn die Zahl durch 3 teilbar, dann Wort Fizz anstatt Zahl ausgeben
3. wenn die Zahl durch 5 teilbar, dann Wort Buzz anstatt Zahl ausgeben
4. wenn die Zahl sowohl durch 3 als auch durch 5 teilbar ist, dann Wort FizzBuzz ausgeben

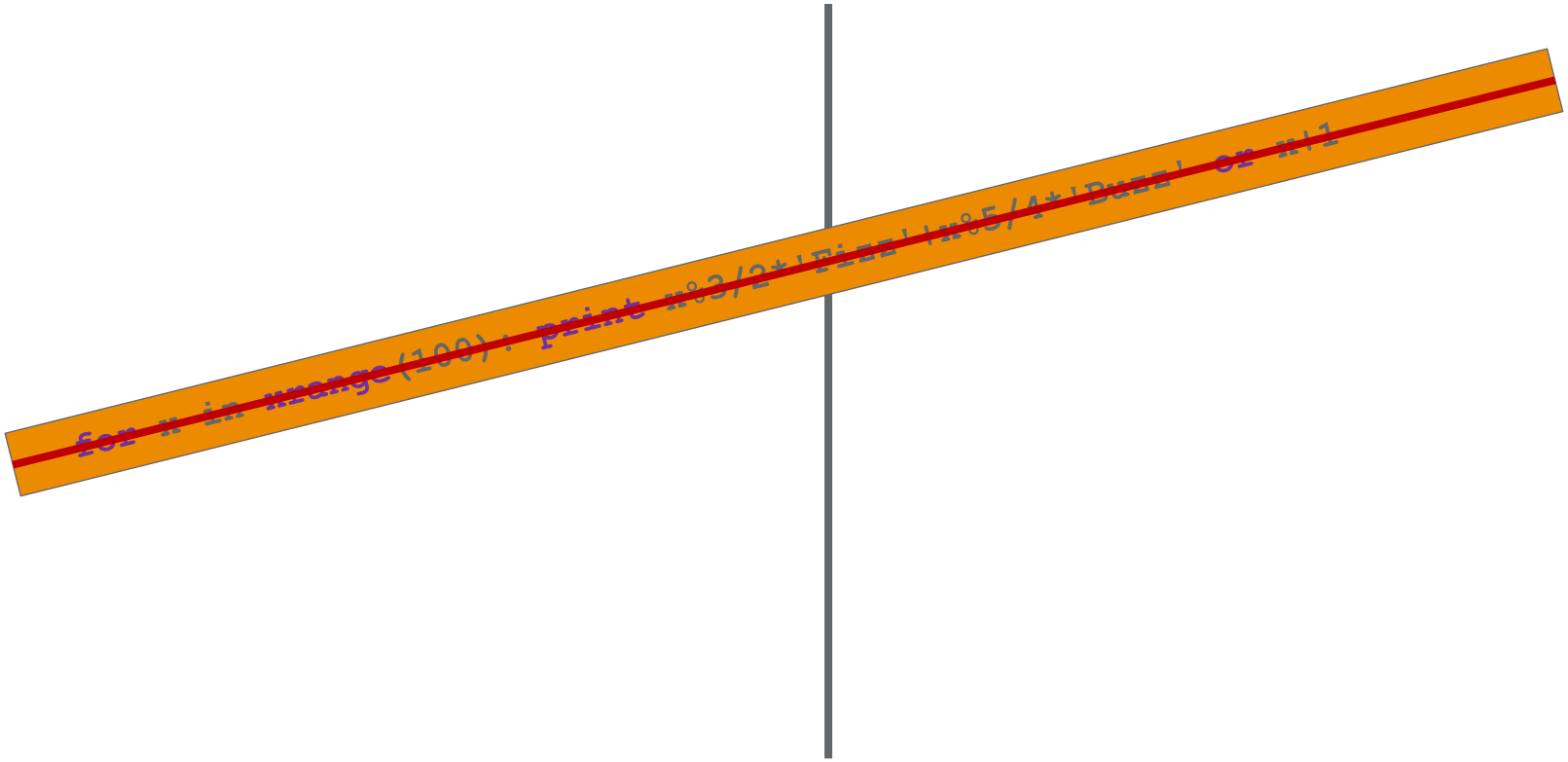


„Philipp! Der Vortrag **MUSS** auch Python Beispiele enthalten!“

tracetronic
GUIDING YOUR VISION

45 Code-Katas: FizzBuzz (Einstiegs-Kata)

1. alle Zahlen von 1 bis 100 ausgeben
2. wenn die Zahl durch 3 teilbar, dann Wort Fizz anstatt Zahl ausgeben
3. wenn die Zahl durch 5 teilbar, dann Wort Buzz anstatt Zahl ausgeben
4. wenn die Zahl sowohl durch 3 als auch durch 5 teilbar ist, dann Wort FizzBuzz ausgeben



Coding-Dojos

tracetronic
GUIDING YOUR VISION

Mehrere Entwickler finden sich zusammen um voneinander zu lernen

Beschäftigung:
- mit Code-Katas
- mit anderen Aufgaben
- sogar gemeinsames Programmieren am Produktivcode

Bewährte Regeln zur Durchführung:
- Pairprogramming an der Aufgabe – andere Entwickler folgen dem Geschehen
- aller 5-10min Rotationsprinzip bei einem der Pair-Partner
- Coding-Dojo Zeit sollte zwischen 30min bis 2Stunden liegen
- am Ende sollte eine Retrospektive stattfinden

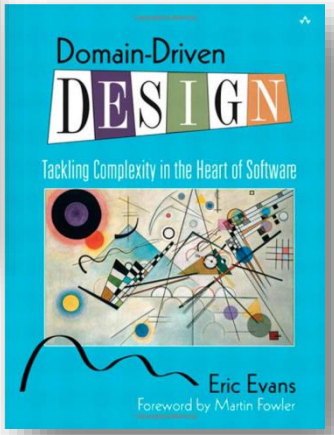
Katas und Dojos in Scrum
- Team reserviert ein Budget
- 90min für Coding-Dojo für einen 2wöchigen Sprint
- 30min für eine wöchentliche Code-Kata

3% der Entwicklerarbeitszeit
- Steigerung der Softwarequalität
- Verbesserung der Programmier-fähigkeiten
investiert.

Empfehlenswertes zum Thema

tracetronic
GUIDING YOUR VISION

Agile Entwicklungspraktiken mit Scrum



Domain-Driven Design

Testgetriebene Entwicklung mit JUnit & FIT

Quellen

Zu Vertiefung des Gebietes kann ich nur das folgende Buch empfehlen, welches auch die Hauptinspirationsquelle für diesen Vortrag war.

Agile Entwicklungspraktiken mit Scrum

Verlag: Dpunkt Verlag
Sprache: Deutsch
ISBN-10: 3898647196
ISBN-13: 978-3898647199

Fragen/Feedback/Retrospektive?

Quelle: https://www.verwaltung.fh-koeln.de/imperia/md/images/dez5/sg52/termintipps/2012/raupe_schmetterling.jpg

tracetronic

GUIDING YOUR VISION

Bildquellen

Quelle:
http://t2.ftcdn.net/jpg/00/45/22/29/400_F_45222981_J5S4wnYYx9uL3XulE34BHcQPY7amXB6g.jpg

Quelle:
http://t2.ftcdn.net/jpg/00/45/22/29/400_F_45222981_J5S4wnYYx9uL3XulE34BHcQPY7amXB6g.jpg

Quelle:
http://1.bp.blogspot.com/__KJX2vsWUNU/TUgiosYABJI/AAAAAAAAAA-c/yj_0pqghoiA/s200/spock-spock-star-trek-smiley-emoticon-000554-large.gif

Quelle:
http://images.zaazu.com/img/desert-storm-desert-storm-soldier-war-smiley-emoticon-000131-large.gif

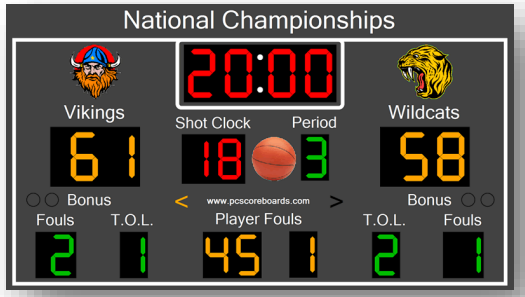
Quelle:
http://frendsbeauty.files.wordpress.com/2011/11/crazy-crazy-mad-straight-jacket-smiley-emoticon-000187-large.gif

Quelle:
http://rlv.zcache.com/mean_yellow_face_smiley_sticker-p217685732820601112836x_325.jpg

Quelle:
http://projectputthatcookiedownnow.com/wp-content/uploads/2011/12/smiley_angry.png

Quelle:
http://cloudfront-assets.reason.com/assets/mc/kmw/2010_06/smiley.jpg

Quelle:
http://image.spreadshirt.net/image-server/v1/designs/15359455,width=178,height=178/Dr.-Smiley.-Mortarboard.-PhD-student.-examination-.png

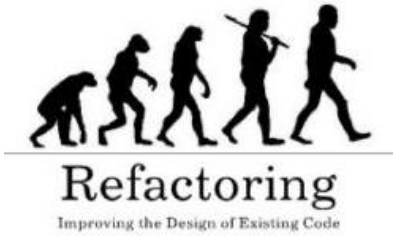


Quelle:
http://www.pcscoreboards.com/basketballscoreboardpro/scoreboardbig.gif

Bildquellen

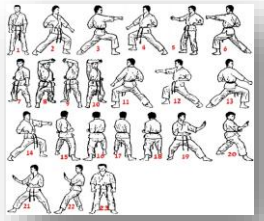
Quelle:
http://bergstr8.blogg.de/files/2012/07/fundament1.jpg

Quelle:
http://bergstr8.blogg.de/files/2012/07/fundament1.jpg



Quelle:
http://www.geekyboy.com/wp-content/uploads/2012/11/evo_refactor.jpg

Quelle:
http://t1.ftcdn.net/jpg/00/28/64/90/400_F_28649094_ZOfgDFqLV5cdAtxPYZJm1Ci5srGXG9Zg.jpg



Quelle:
http://kbl.bplaced.net/images/karate-Dateien/kata/0-Kata-Heian-Shodan.jpg

Quelle:
http://cdn.slidesharecdn.com/ss_thumbnails/codekata-110122075320-phpapp02-thumbnail-4.jpg

Quelle:
http://ebmedia.eventbrite.com/s3-s3/eventlogos/12266441/1422786593-2.png

Quelle:
http://www.fry-it.com/blog/london-python-code-dojo/blogPostImage