



Refactoring mit der Mikado-Methode



Orientation in Objects GmbH

Weinheimer Str. 68
68309 Mannheim

www.oio.de
info@oio.de

Version: 1.0

Abstract

Viele von uns haben tagtäglich mit Legacy-Code zu tun. Mal eben schnell etwas umzubauen, scheitert typischerweise an den fehlenden Tests, zudem ist der Quellcode oft überhaupt schlecht testbar.

In diesem Vortrag wird anhand von praktischen Codebeispielen gezeigt, wie man zunächst ein automatisiertes Sicherheitsnetz aufspannt. Anschließend werden komplexere Refactorings durchgeführt, ohne jedoch zu viele Baustellen gleichzeitig aufzureißen. Die Mikado-Methode hilft dabei, den Überblick zu behalten und in möglichst kleinen und nachvollziehbaren Schritten vorzugehen. Das Ziel ist das Aufbrechen stark gekoppelter Abhängigkeiten, um so neue Tests hinzufügen zu können. Zudem wird der Code besser lesbar sein und lässt sich so auch leichter warten und wiederverwenden.

Falk Sippach (@sipsack)

Trainer, Berater, Entwickler

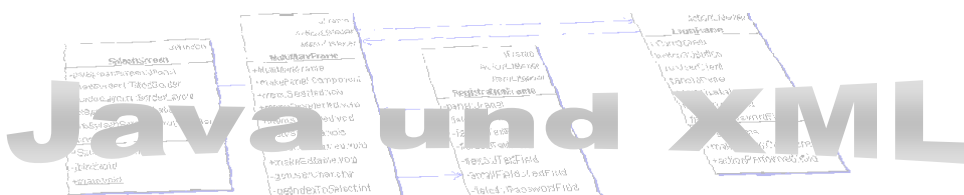


Co-Organisator

Architektur
Agile Softwareentwicklung
Codequalität



Committer DukeCon



) Software Factory)


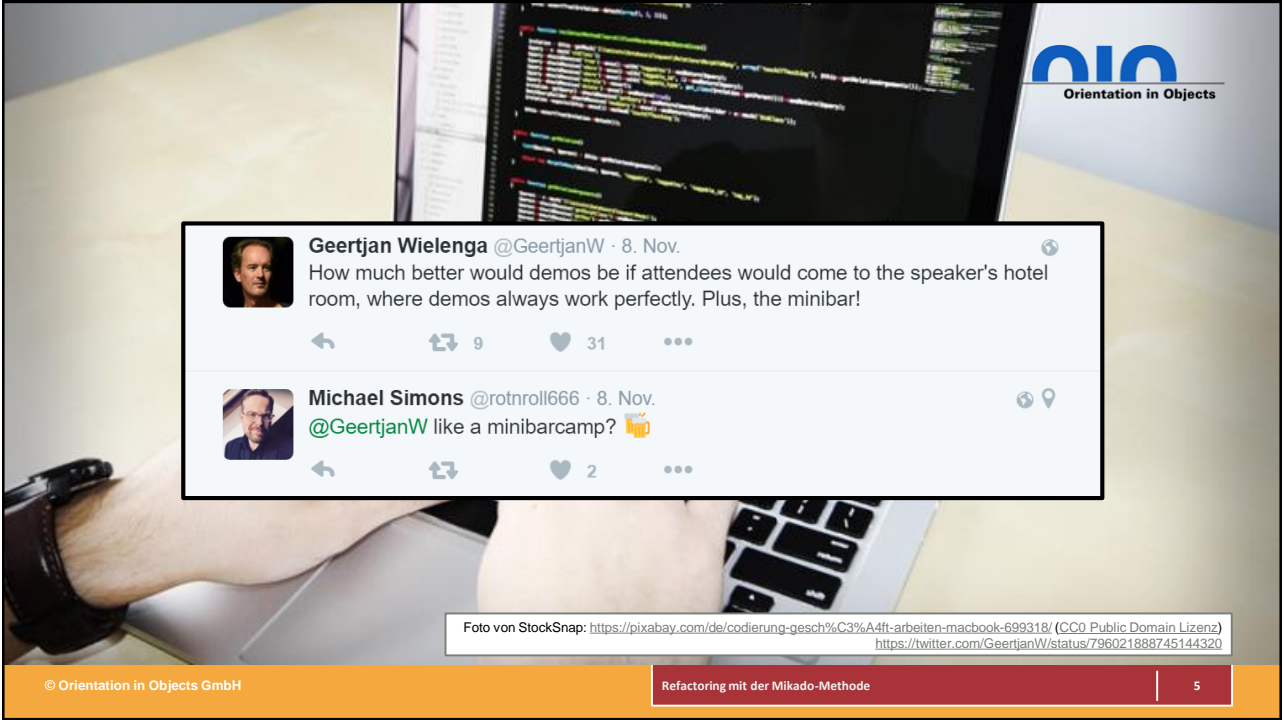
- Schlüsselfertige Realisierung von Java Software
- Individualsoftware
- Pilot- und Migrationsprojekte
- Sanierung von Software
- Software Wartung


) Object Rangers)

- Unterstützung laufender Java Projekte
- Perfect Match
- Rent-a-team
- Coaching on the project
- Inhouse Outsourcing

) Competence Center)

- Schulungen, Coaching, Weiterbildungsberatung, Train & Solve-Programme
- Methoden, Standards und Tools für die Entwicklung von offenen, unternehmensweiten Systemen



**Geertjan Wielenga** @GeertjanW · 8. Nov.
How much better would demos be if attendees would come to the speaker's hotel room, where demos always work perfectly. Plus, the minibar!




**Michael Simons** @rotnroll666 · 8. Nov.
[@GeertjanW](#) like a minibarcamp? 🍷

Foto von StockSnap: <https://pixabay.com/de/codierung-gesch%C3%A4ft-arbeiten-macbook-699318/> (CC0 Public Domain Lizenz)
<https://twitter.com/GeertjanW/status/796021888745144320>

© Orientation in Objects GmbH

Refactoring mit der Mikado-Methode

5



Legacy Code

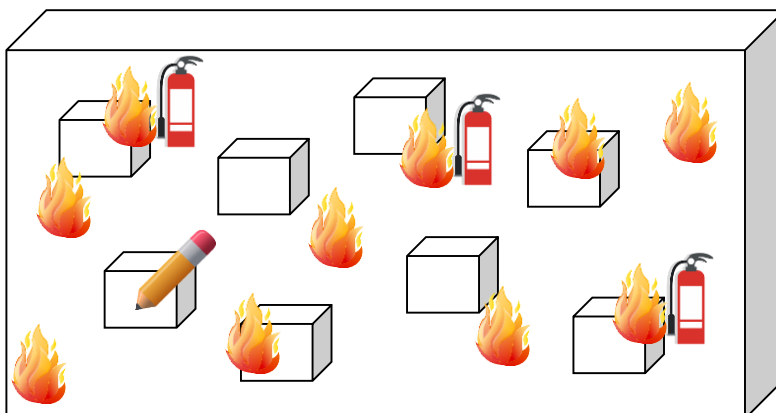
© Orientation in Objects GmbH

Refactoring mit der Mikado-Methode

6

Code ändern? Nur wie?

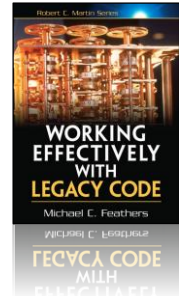
***Brownfield-Projekt. Code von anderen.
Fehlende Dokumentation, kaum Tests.
Änderungen geraten außer Kontrolle.***





Michael Feathers

“ Code
without tests
is bad code.



J. B. Rainsberger

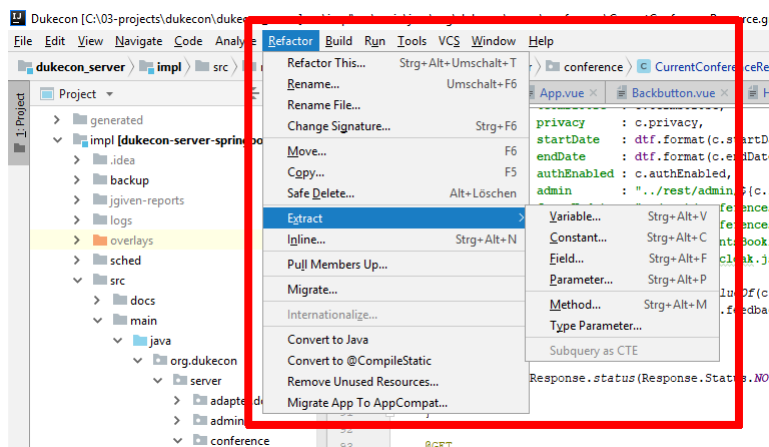
“ Legacy code is **valuable**
code that we feel **afraid to**
change.

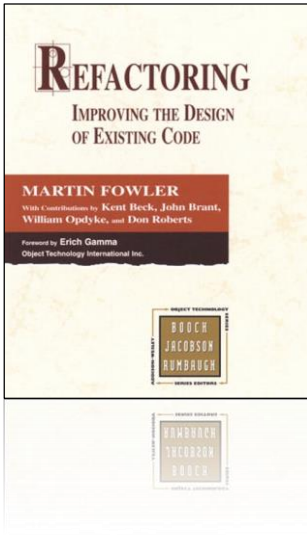


Gründe für Refactoring

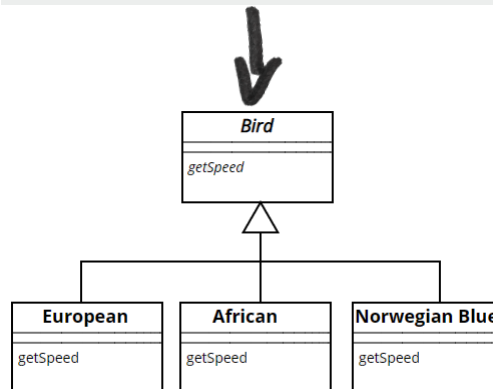
Verstehen
Fehler beheben
Neues Feature
Optimierung

Toolunterstützung bei einfachen Refactorings





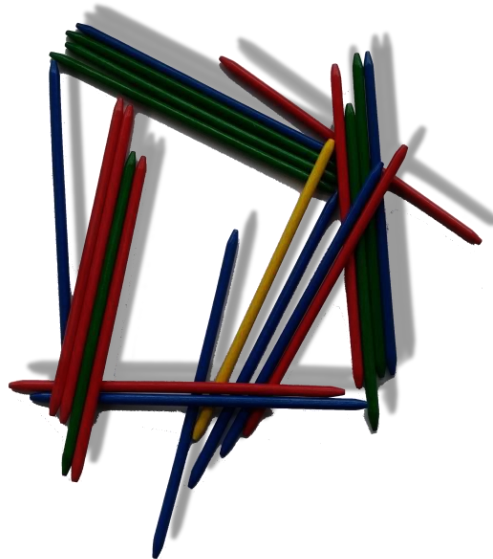
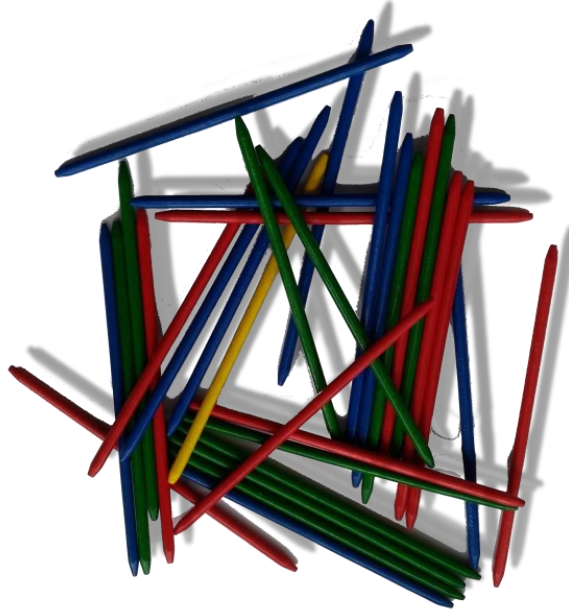
```
double getSpeed() {
    switch (_type) {
        case EUROPEAN:
            return getBaseSpeed();
        case AFRICAN:
            return getBaseSpeed() - getLoadFactor() * _numberOfCoconuts;
        case NORWEGIAN_BLUE:
            return (_isNailed) ? 0 : getBaseSpeed(_voltage);
    }
    throw new RuntimeException ("Should be unreachable");
}
```

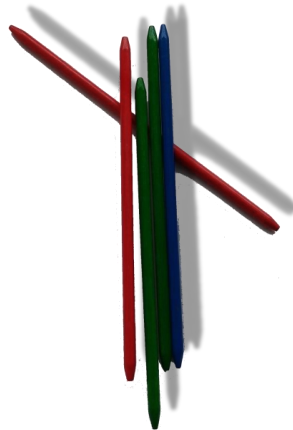
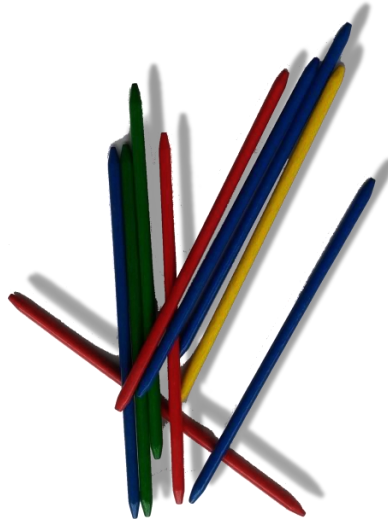


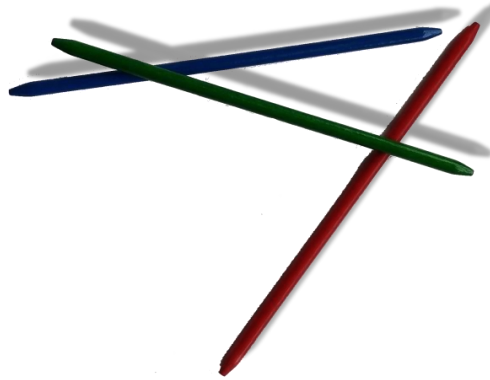
The **Mikado Method** is a **structured way** to make **significant changes** to complex code.

... for **performing changes** to a **system that's too large for analyze-then-edit**, which means basically **any production system** in the world.

(Ola Ellnestam, Daniel Brolund: "The Mikado Method")



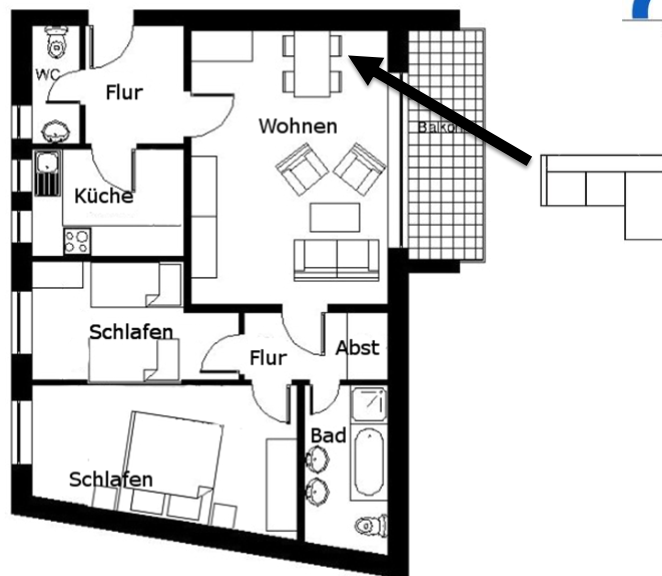


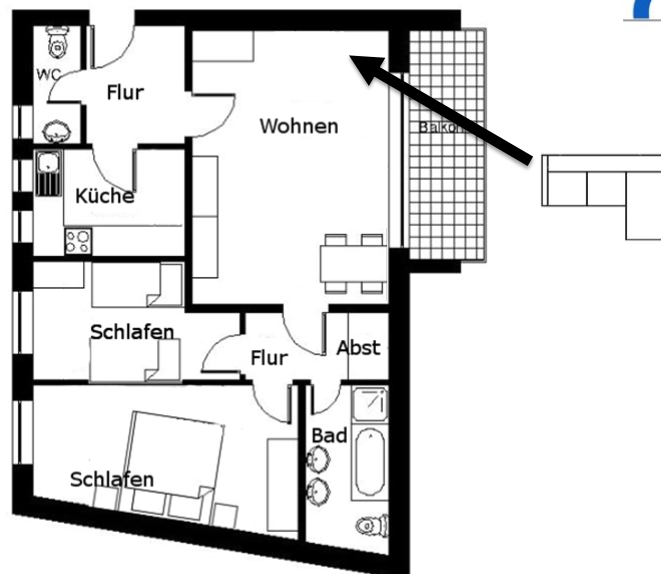
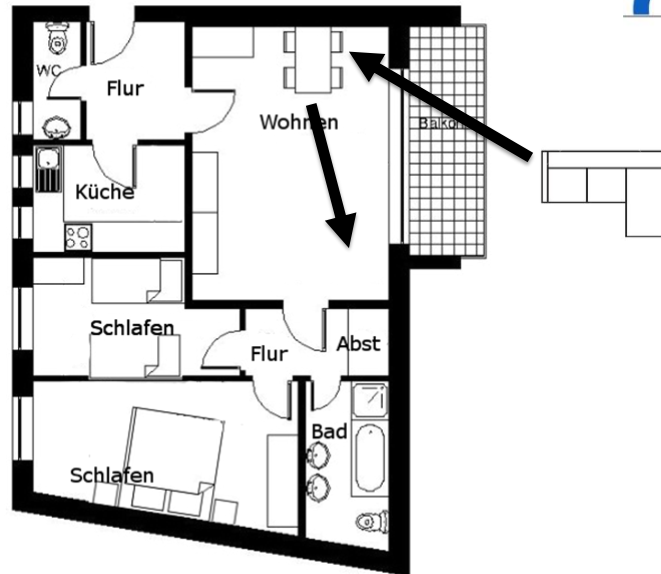


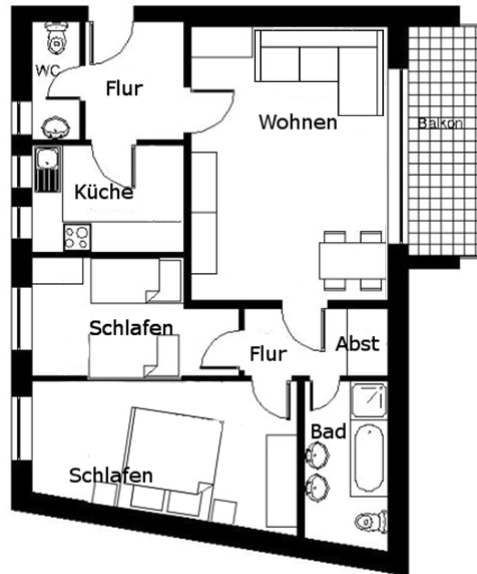
Was ist die Mikado-Methode?

Mikado Methode für Dummies

Möbel rücken



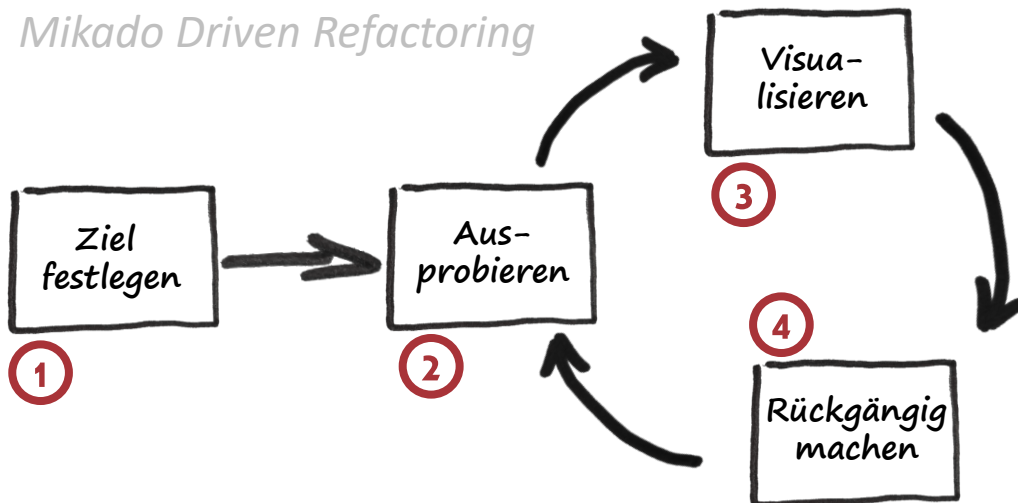




The Mikado Method can help you **visualize, plan, and perform** business value–focused **improvements** over several iterations and increments of work, **without** ever having **a broken codebase** during the process.

(Ola Ellnestam, Daniel Brolund: "The Mikado Method")

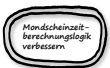
Mikado Driven Refactoring



1

Ziel festlegen

=



***Startpunkt der Änderung.
Erfolgskriterium für Ende.***

2

Ausprobieren

=

***Experimentieren.
Hypothesen prüfen.
Sehen, was kaputt geht.***

3

Visualisieren

=



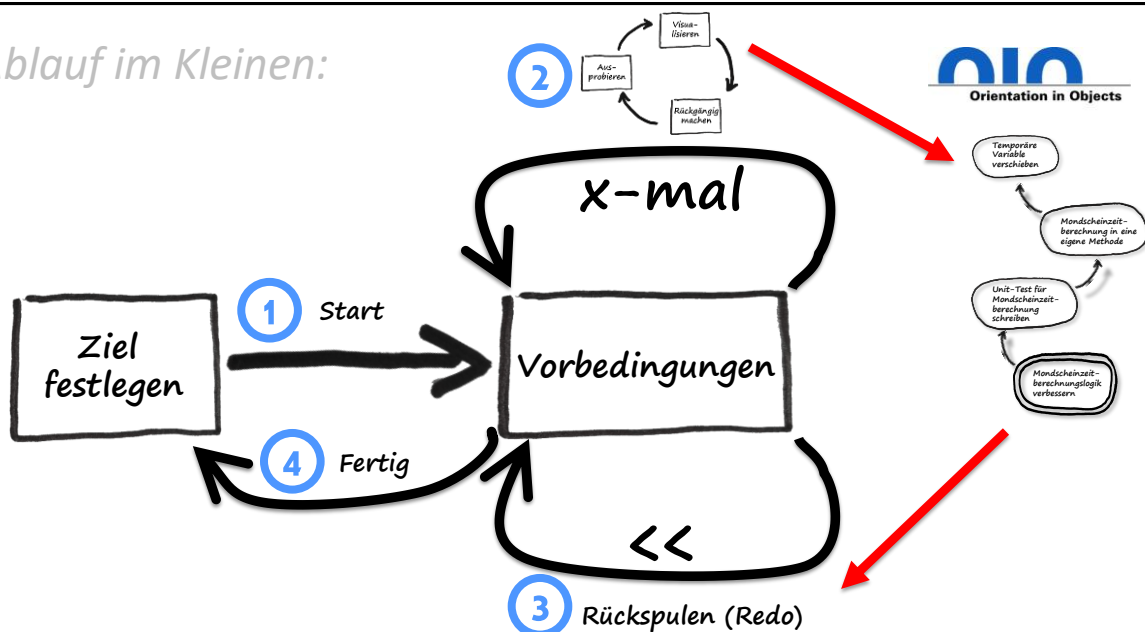
***Ziel und notwendige
Vorbedingungen aufschreiben.
Mikado Graph erstellen.***

Rückgängig machen

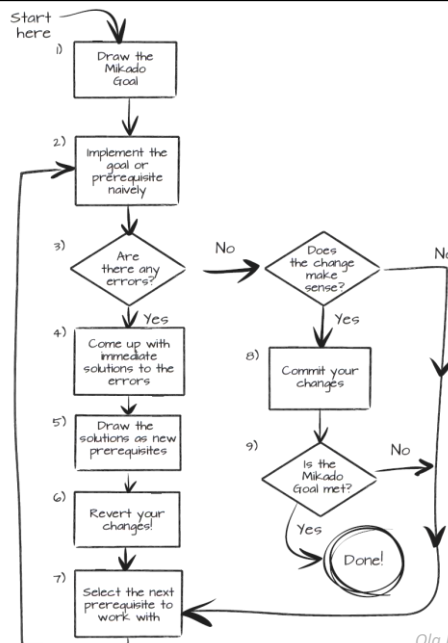
=

**Vorherigen funktionierenden
Stand wiederherstellen.**

Ablauf im Kleinen:



Ablauf im Großen:

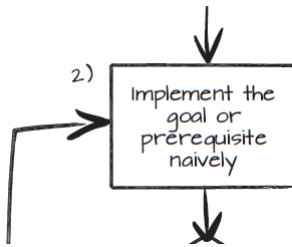


Ola Ellnestam, Daniel Brolund: "The Mikado Method"

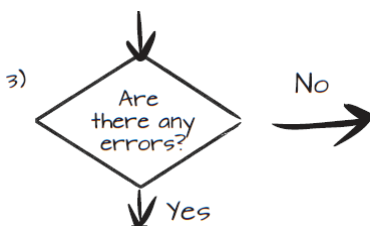


Konkrete Aufgabe, z. B. User Story.
Auf Papier schreiben.

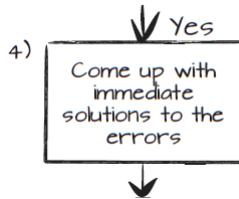




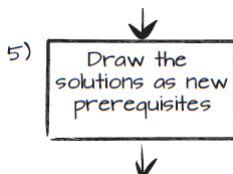
***Direkt eine einfache Lösung ausprobieren.
Experimentieren statt analysieren.
Hilfe durch Compiler und Tests.***



***Auftretende Fehler haben die Ursache in weiteren
Abhängigkeiten.
Zu Schritt 8, wenn keine Fehler.
Vorsicht bei Laufzeitfehlern (Nullpointer).***

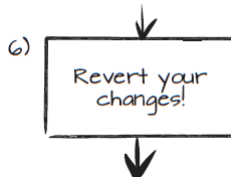


***Direkt sofortige Lösungen zu den Fehlern finden.
Überanalysieren vermeiden.
Weiter zugrundeliegende Einschränkungen
werden in späteren Iterationen behandelt.***

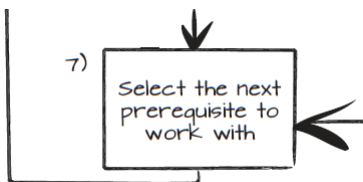


***Lösung aufzeichnen und mit Vorgänger verbinden.
Wissen über das System aufbauen.***



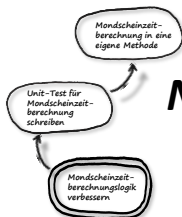


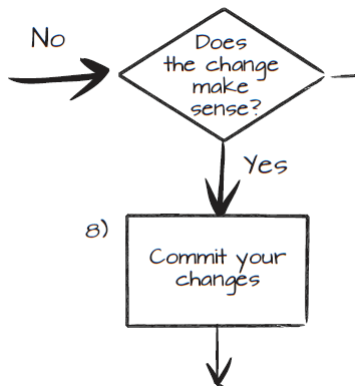
***Bei Fehlern immer zurückrollen.
Weitere Bearbeitung auf kaputten Zustand sehr
fehleranfällig.
Zurückgerollte Informationen stecken im Graph.***



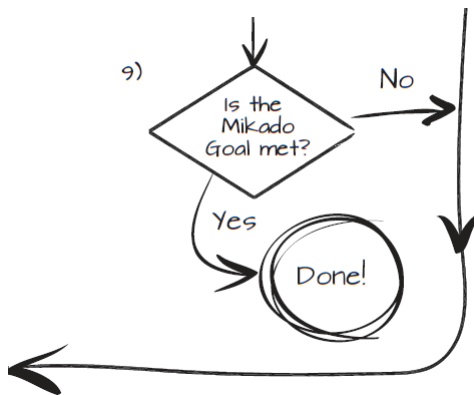
***Nächste Vorbedingung auswählen und zu Schritt 2.
Immer mit sauberen System starten.***

Nächste Vorbedingung naiv implementieren ...





***Vorbedingung abhaken, wenn keine Fehler mehr.
Committen, wenn es Sinn ergibt.***



***Mikado-Goal und alle Vorbedingungen erfüllt?
Fertig!***

```
package de.oio.refactoring.badtelefon;

public class Kunde {
    double gebuehr = 0.0;
    Tarif tarif;

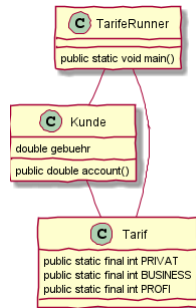
    public Kunde(int tarifArt) {
        this.tarif = new Tarif(tarifArt);
    }

    public void account(int minuten, int stunde, int minute)
    {
        boolean mondschein = false;
        double preis = 0;

        // Mondscheinzeit ?
        if (stunde < 9 || stunde > 18)
            mondschein = true;

        // Gesprachspreis ermitteln
        switch (tarif.tarif) {
            case Tarif.PRIVAT:
                [...]
            }
        gebuehr += preis;
    }

    public double getGebuehr() {
        return gebuehr;
    }
}
```



```
package de.oio.refactoring.badtelefon;

import java.util.Arrays;
import java.util.Random;

public class TarifeRunner {
    public static void main(String args[]) {
        Random random = new Random();
        for(Integer tarif : Arrays.asList(Tarif.PRIVAT, Tarif.BUSINESS, Tarif.PROFI)) {
            System.out.println(String.format("\nVerarbeitung von Tarif %s", tarif));
            Kunde k = new Kunde(tarif);
            [...]
        }
    }
}
```

```
package de.oio.refactoring.badtelefon;

public class Tarif {
    public final static int PRIVAT = 0;
    public final static int BUSINESS = 1;
    public final static int PROFI = 2;

    int tarif = 0;

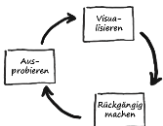
    public Tarif(int tarif) {
        this.tarif = tarif;
    }
}
```

Temporäre
Variable
verschieben

Mondscheinzeit-
berechnung in eine
eigene Methode

Unit-Test für
Mondscheinzeit-
berechnung
schreiben

Mondscheinzeit-
berechnungslogik
verbessern




Zurückspulen
Experimentieren

Zurückrollen
Visualisieren
Experimentieren

Zurückrollen
Visualisieren
Experimentieren

Zurückrollen
Visualisieren
Experimentieren
Mikado-Ziel festlegen





Orientation in Objects

```
classDiagram
    class TarifeRunner {
        +public static void main()
    }
    class Kunde {
        +double gebuehr
        +public double account()
    }
    class Tarif {
        +public static final int PRIVAT
        +public static final int BUSINESS
        +public static final int PROFI
    }
    TarifeRunner --> Kunde
    Kunde --> Tarif
```

Foto von StockSnap: <https://pixabay.com/de/codierung-gesch%C3%A4ft-arbeiten-macbook-699318/> (CC0 Public Domain Lizenz)

© Orientation in Objects GmbH

Refactoring mit der Mikado-Methode

45





Orientation in Objects

© Orientation in Objects GmbH

Refactoring mit der Mikado-Methode

46



Großes Beispiel

Einfach loslegen?

Es fehlen automatisierte Tests!
Machen wir auch nichts kaputt?


```
# suppose that our legacy code is this program called 'game'
$ game > GOLDEN_MASTER

# after some changes we can check to see if behaviour has changed
$ game > OUT-01
$ diff GOLDEN_MASTER OUT-01
# GOLDEN_MASTER and OUT-01 are the same

# after some other changes we check again and...
$ game > OUT-02
$ diff GOLDEN_MASTER OUT-02
# GOLDEN_MASTER and OUT-02 are different -> behaviour changed
```




Golden Master (aka characterization tests)

```
@Before
public void init() {
    originalSysOut = System.out;
    consoleStream = new ByteArrayOutputStream();
    PrintStream printStream = new PrintStream(consoleStream);
    System.setOut(printStream);
}

@Test
public void testSimpleOutput() {
    System.out.println("Hallo Publikum!");
    System.out.print("Hallo Falk!");
    assertEquals("Hallo Publikum!\r\nHallo Falk!", consoleStream.toString());
}


@After
public void teardown() {
    System.setOut(originalSysOut);
}
```


1



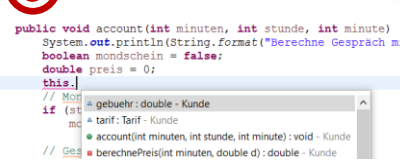
```
public static void main(String... args) throws Exception {
    WebDriver driver = new FirefoxDriver();
    driver.get("http://www.retest.de");
    while (true) {
        List<WebElement> links = driver.findElements(By.tagName("a"));
        links.get(random.nextInt(links.size())).click();
        Thread.sleep(500);
        List<WebElement> fields =
            driver.findElements(By.xpath("//input[@type='text']"));
        WebElement field = fields.get(random.nextInt(fields.size()));
        field.sendKeys(randomString());
        Thread.sleep(500);
    }
}
```

2






3



4




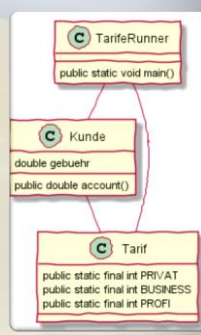
https://entwicklertag.de/frankfurt/2016/sites/entwicklertag.de/frankfurt.2016/files/slides/Bei%20Uns%20testen%20lauter%20Affen_0.pdf

© Orientation in Objects GmbH

Refactoring mit der Mikado-Methode

51





```
classDiagram
    class TariffeRunner {
        +public static void main()
    }
    class Kunde {
        +double gebuehr
        +public double account()
    }
    class Tarif {
        +public static final int PRIVAT
        +public static final int BUSINESS
        +public static final int PROFI
    }
    TariffeRunner -- Kunde
    Kunde -- Tarif
```

Foto von StockSnap: <https://pixabay.com/de/codierung-gesch%C3%A4ft-arbeiten-macbook-699318/> (CC0 Public Domain Lizenz)

© Orientation in Objects GmbH

Refactoring mit der Mikado-Methode

52

26



Fazit

Vorteile

Stabiles System trotz Änderungen.

***Bessere Kommunikation und
Zusammenarbeit.***

Leichtgewichtig und fokussiert.

Wann nutzen?

- ① ***Architektur im Betrieb verbessern.***
- ② ***Brownfield Entwicklung.***
- ③ ***Refactoring Projekt.***

***In kleinen Schritten verbessern.
Im gleichen Branch neue Features
kontinuierlich ausliefern.***

***Häufige Situation.
Existierende Anwendungen
verbessern.***

Langdauernde Verbesserung. Eigener Branch.

Angst vorm Revert:

***Reverting scheint wie Wegwerfen.
Aber Mikado-Graph enthält Infos.
Wissensaufbau/-transfer über
System, Domäne, Technologie.***

Arbeitsmittel:

①

Whiteboard/Flipchart/Papier

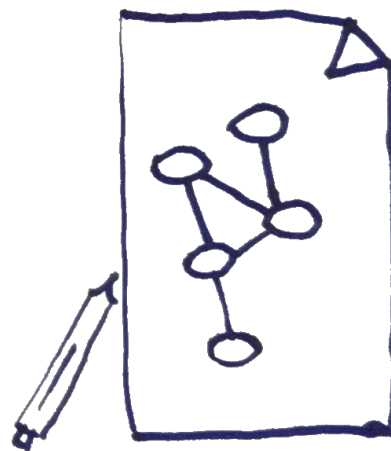
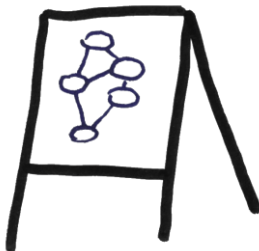
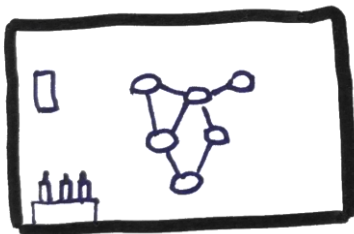
②

Mindmap

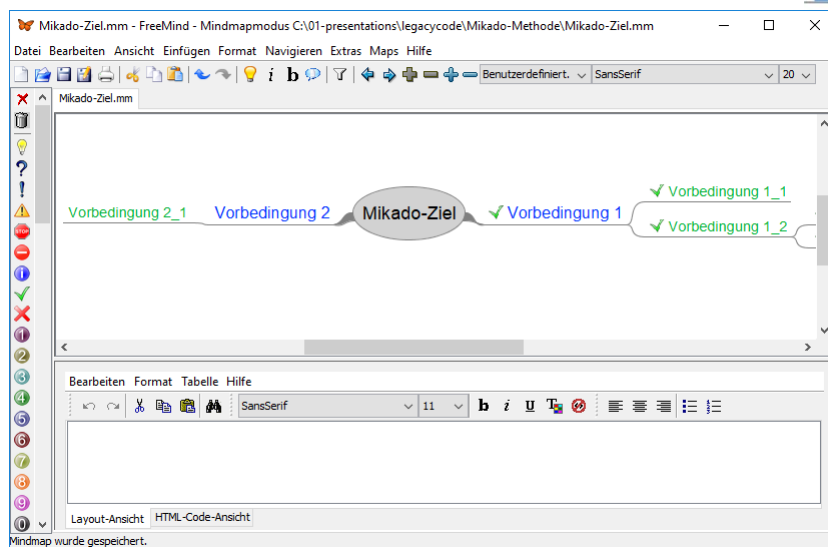
③

Visio, yEd

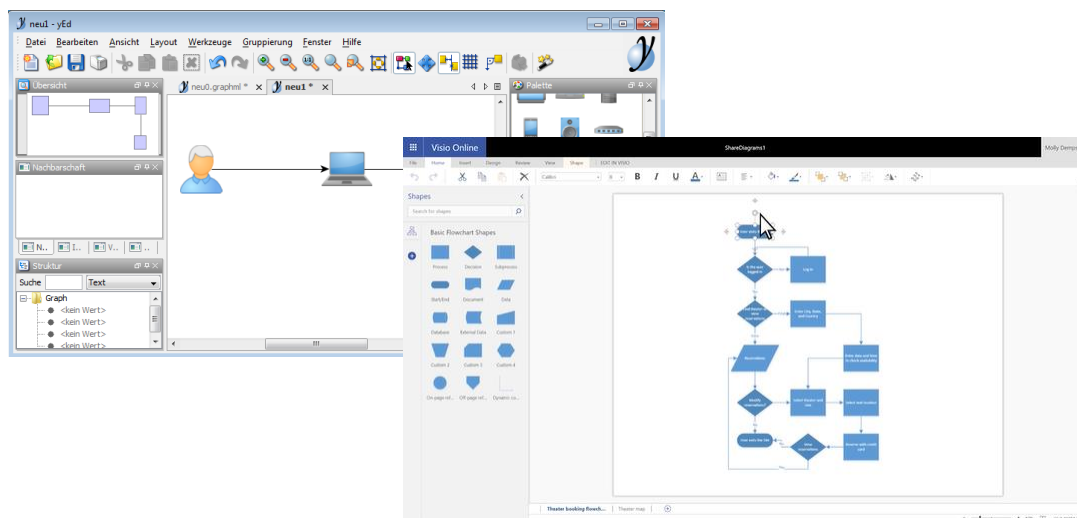
①

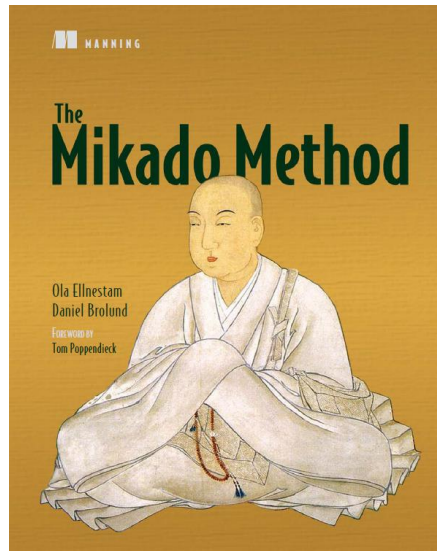


2



3





Vielen Dank für Ihre Aufmerksamkeit!

Orientation in Objects GmbH

Weinheimer Str. 68
68309 Mannheim

www.oio.de
info@oio.de