Orientation in Objects

Refactoring mit der Mikado-Methode

Orientation in Objects GmbH

Weinheimer Str. 68
68309 Mannheim

www.oio.de
info@oio.de

KARLSRUHER ENTWICKLERTAG

Version: 1.0

Abstract

Orientation in Objects

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.

Ihr Sprecher

Orientation in Objects

Falk Sippach (@sippsack)

Trainer, Berater, Entwickler

Co-Organisator

Commiter DukeCon

Architektur
Agile Softwareentwicklung
Codequalität

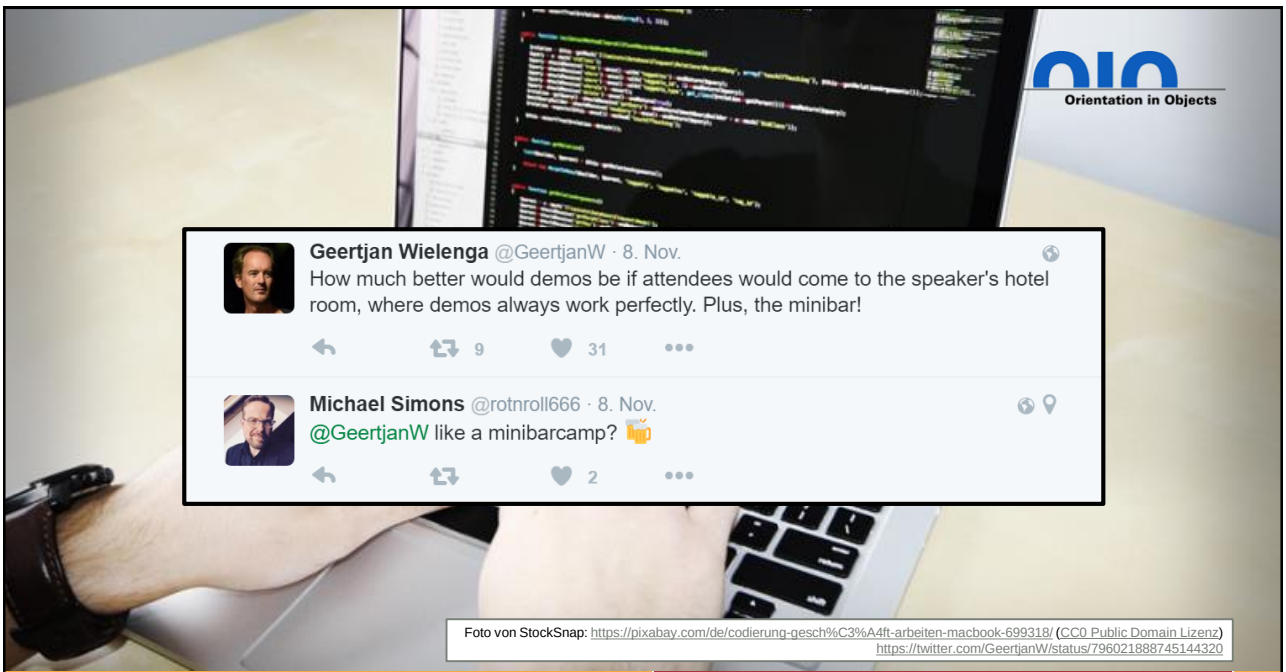


Foto von StockSnap: https://pixabay.com/de/codierung-gesch%C3%A4ft-arbeiten-macbook-699318/ (CC0 Public Domain Lizenz)
https://twitter.com/GeertjanW/status/796021888745144320

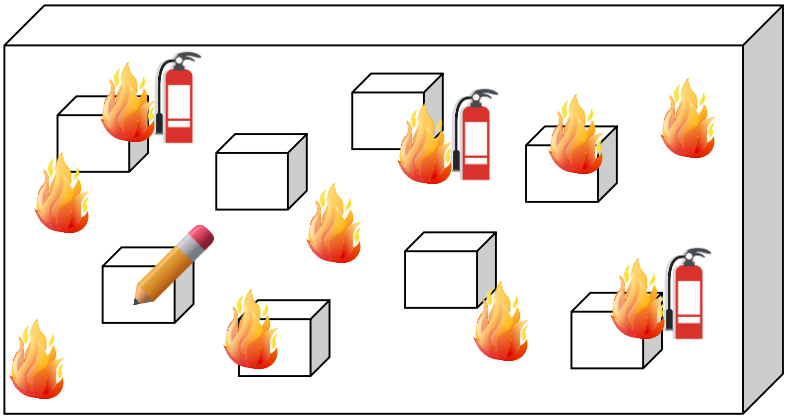
Orientation in Objects

Legacy Code

Orientation in Objects

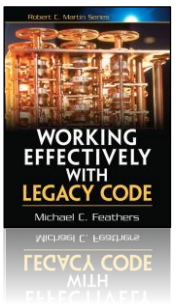
Code ändern? Nur wie?

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



“ Code **without tests** is **bad code**.

Michael Feathers



J. B. Rainsberger

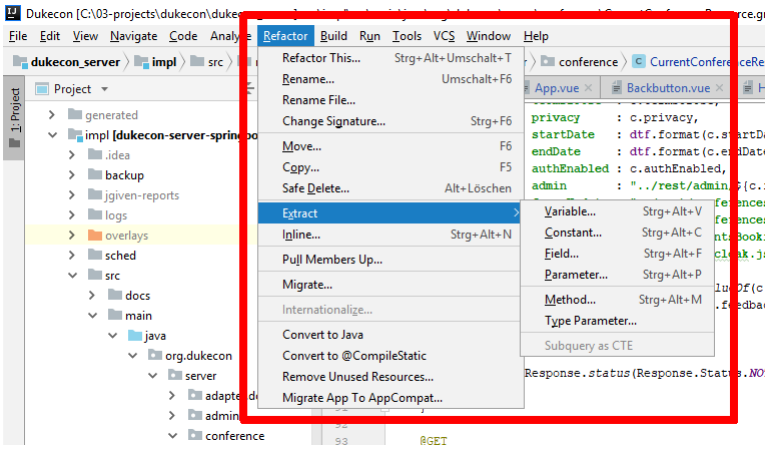
> “ Legacy code is **valuable code** that we feel **afraid to change**.

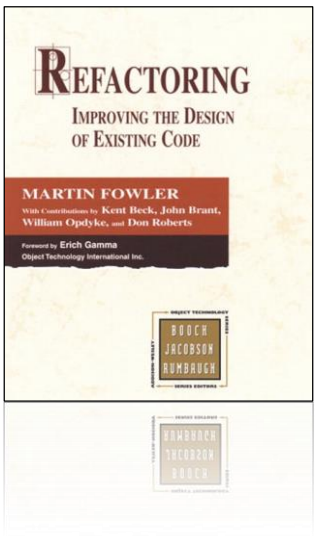
Foto von PublicDomainPictures, CC0 Public Domain Lizenz, https://pixabay.com/de/menschen-abdeckung-schrei-314481/

Gründe für Refactoring

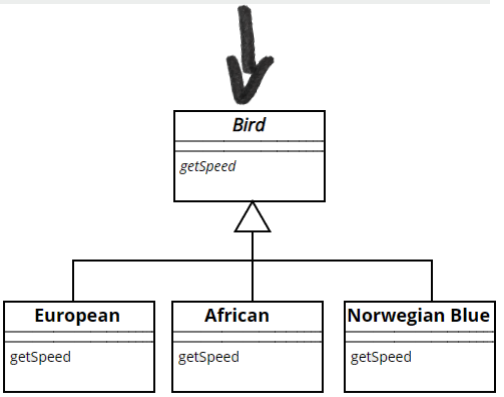
Verstehen
Fehler beheben
Neues Feature
Optimierung

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");
}
```



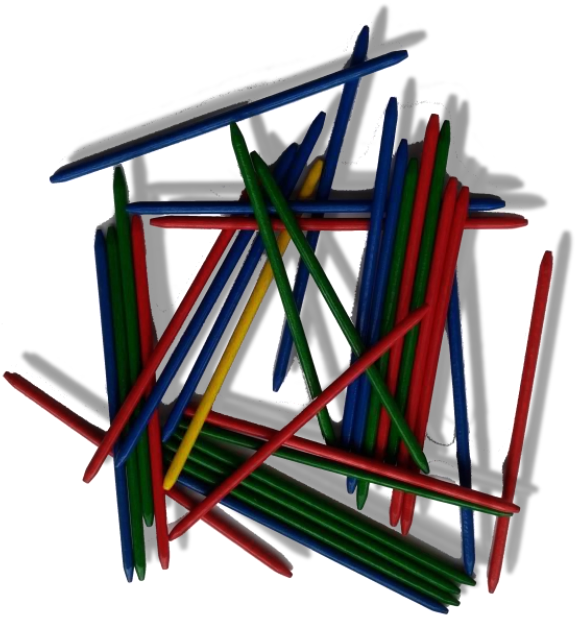
Orientation in Objects

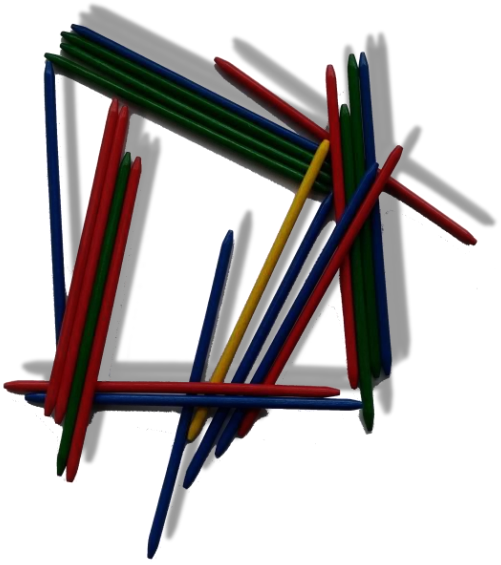
*The **Mikado Method** is a **structured way** to make **significant changes** to **complex code**.*

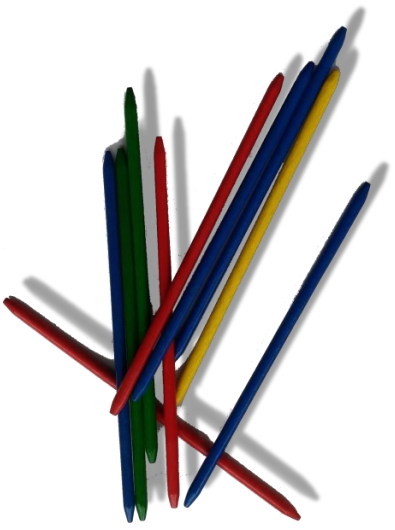
*... for **performing changes** to a **system that's too large for analyze-then-edit**, which means basically **any production system** in the world.*

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

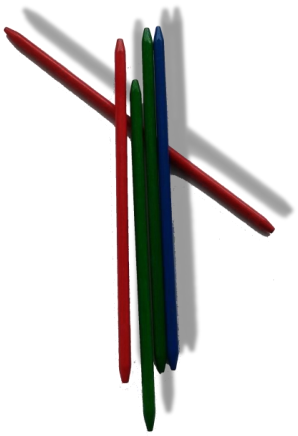
Orientation in Objects



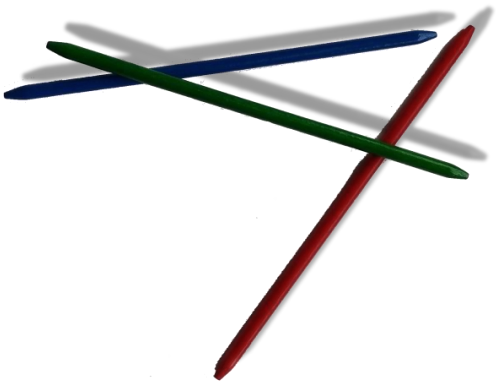




Orientation in Objects



Orientation in Objects



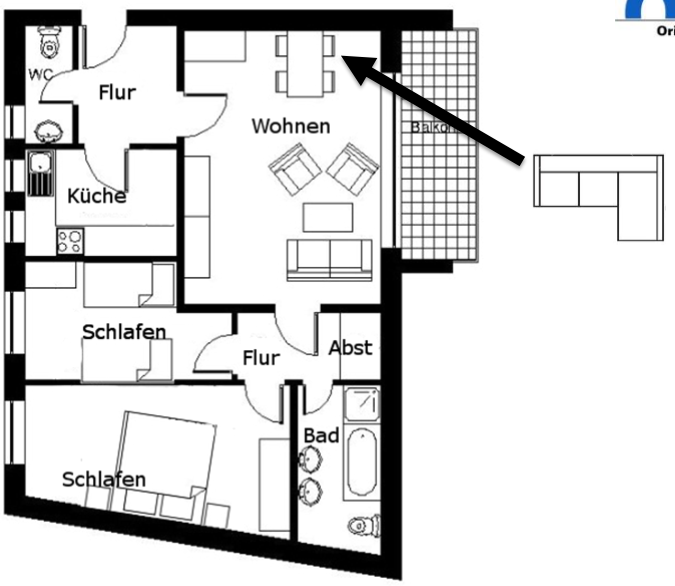
Orientation in Objects

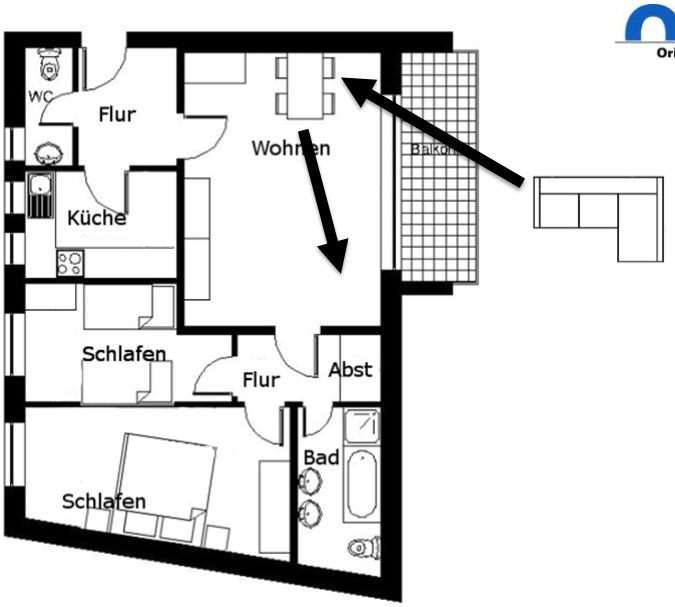
Was ist die Mikado-Methode?

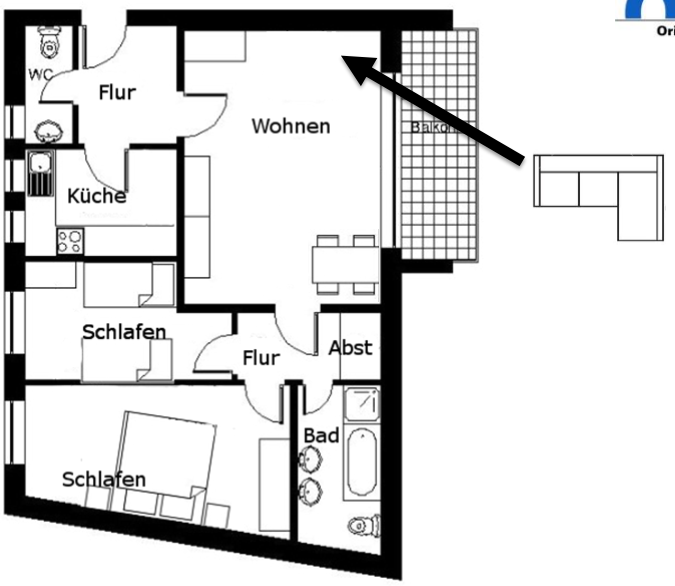
Orientation in Objects

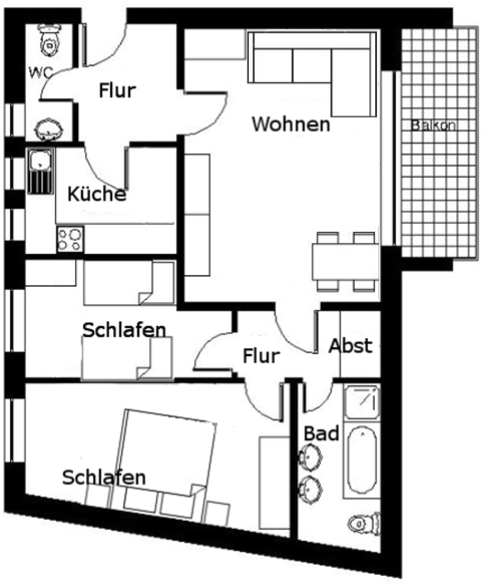
Mikado Methode für Dummies

Möbel rücken


WC
Flur
Wohnen
Küche
Schlafen
Flur
Abst
Bad
Schlafen


WC
Flur
Küche
Schlafen
Flur
Abst
Bad
Schlafen

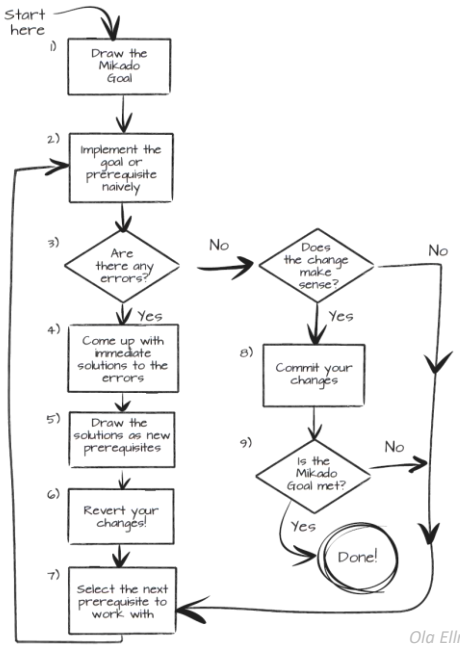

WC
Flur
Wohnen
Balkon
Küche
Schlafen
Flur
Abst
Bad
Schlafen


WC
Flur
Wohnen
Balkon
Küche
Schlafen
Flur
Abst
Bad
Schlafen

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")

Ablauf im Großen:

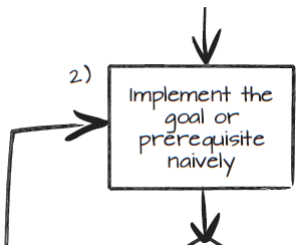


Ola Ellnestam, Daniel Brolund: "The Mikado Method"

Orientation in Objects

Konkrete Aufgabe, z. B. User Story.

Auf Papier schreiben.

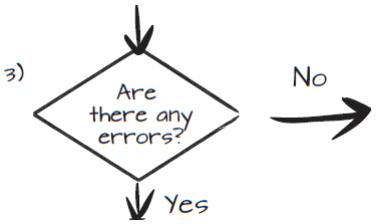


Orientation in Objects

Direkt eine einfache Lösung ausprobieren.

Experimentieren statt analyisieren.

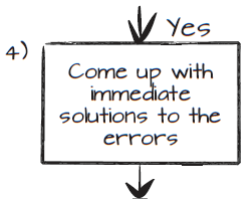
Hilfe durch Compiler und Tests.



Auftretende Fehler haben die Ursache in weiteren Abhängigkeiten.

Zu Schritt 8, wenn keine Fehler.

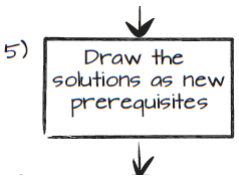
Vorsicht bei Laufzeitfehlern (Nullpointer).



Direkt sofortige Lösungen zu den Fehlern finden.

Überanalyisieren vermeiden.

Weiter zugrundeliegende Einschränkungen werden in späteren Iterationen behandelt.



Orientation in Objects

Lösung aufzeichnen und mit Vorgänger verbinden.

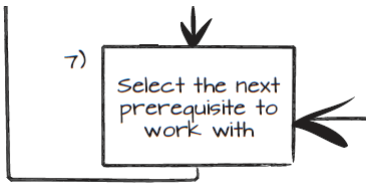
Wissen über das System aufbauen.

Orientation in Objects

Bei Fehlern immer zurückrollen.

Weitere Bearbeitung auf kaputten Zustand sehr fehleranfällig.

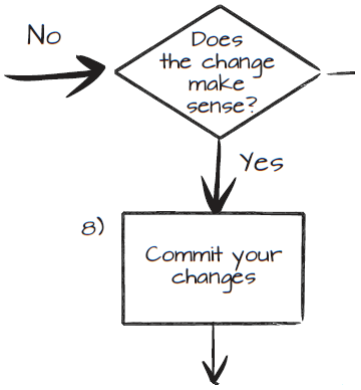
Zurückgerollte Informationen stecken im Graph.



Nächste Vorbedingung auswählen und zu Schritt 2.

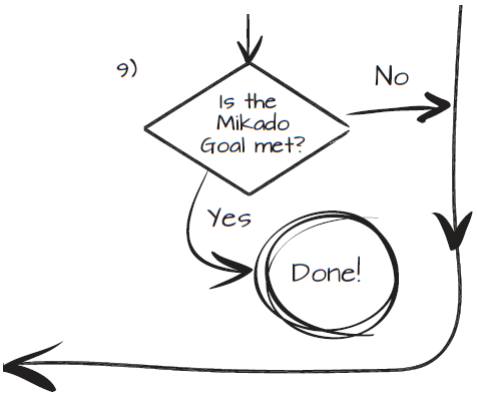
Immer mit sauberen System starten.

Nächste Vorbedingung naiv implementieren …



Vorbedingung abhaken, wenn keine Fehler mehr.

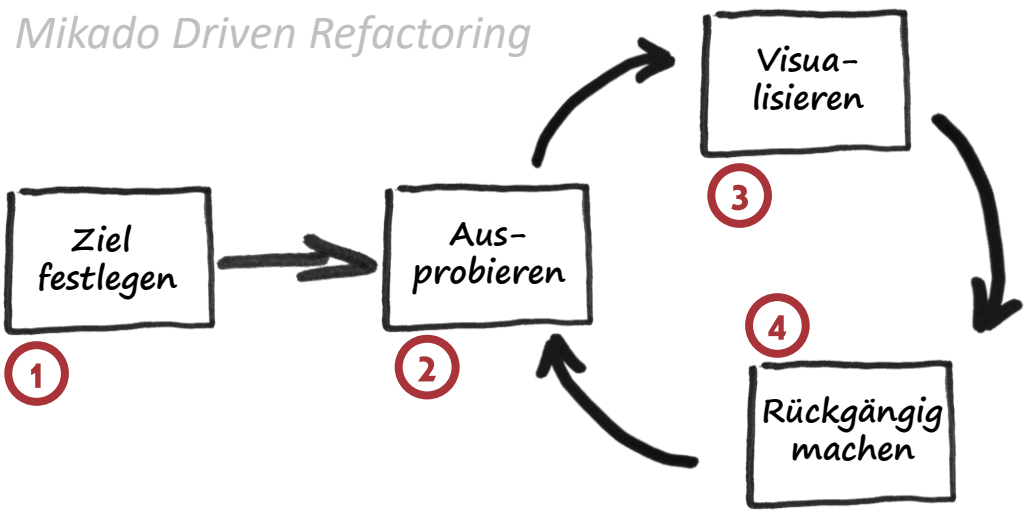
Committen, wenn es Sinn ergibt.



Mikado-Goal und alle Vorbedingungen erfüllt?

Fertig!

Mikado Driven Refactoring



1

Orientation in Objects

Ziel festlegen

=

***Startpunkt** der Änderung.*

***Erfolgskriterium** für Ende.*

2

Orientation in Objects

Ausprobieren

=

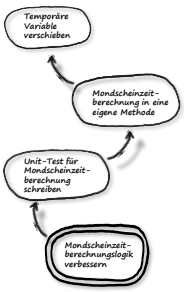
Experimentieren.

***Hypothesen** prüfen.*

*Sehen, was **kaputt geht.***

3

Orientation in Objects



Visualisieren

=

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

4

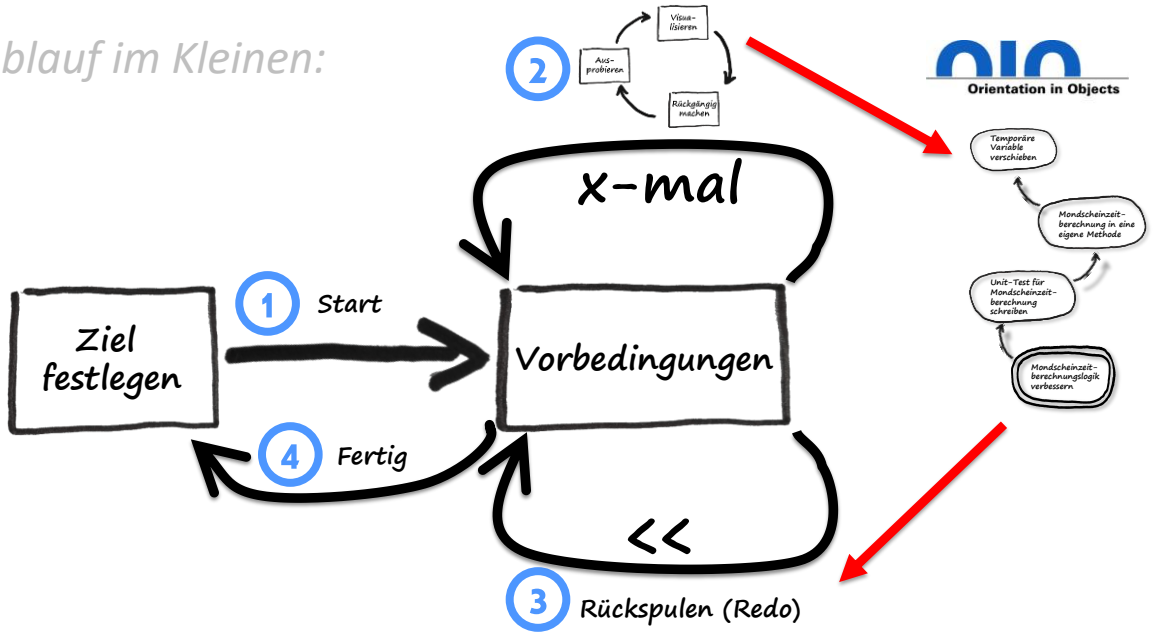
Orientation in Objects

Rückgängig machen

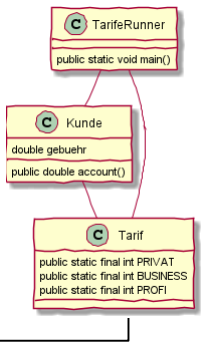
=

***Vorherigen** funktionierenden*
*Stand **wiederherstellen.***

Ablauf im Kleinen:




```
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;

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

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



Orientation in Objects

```
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;
     }
}
```

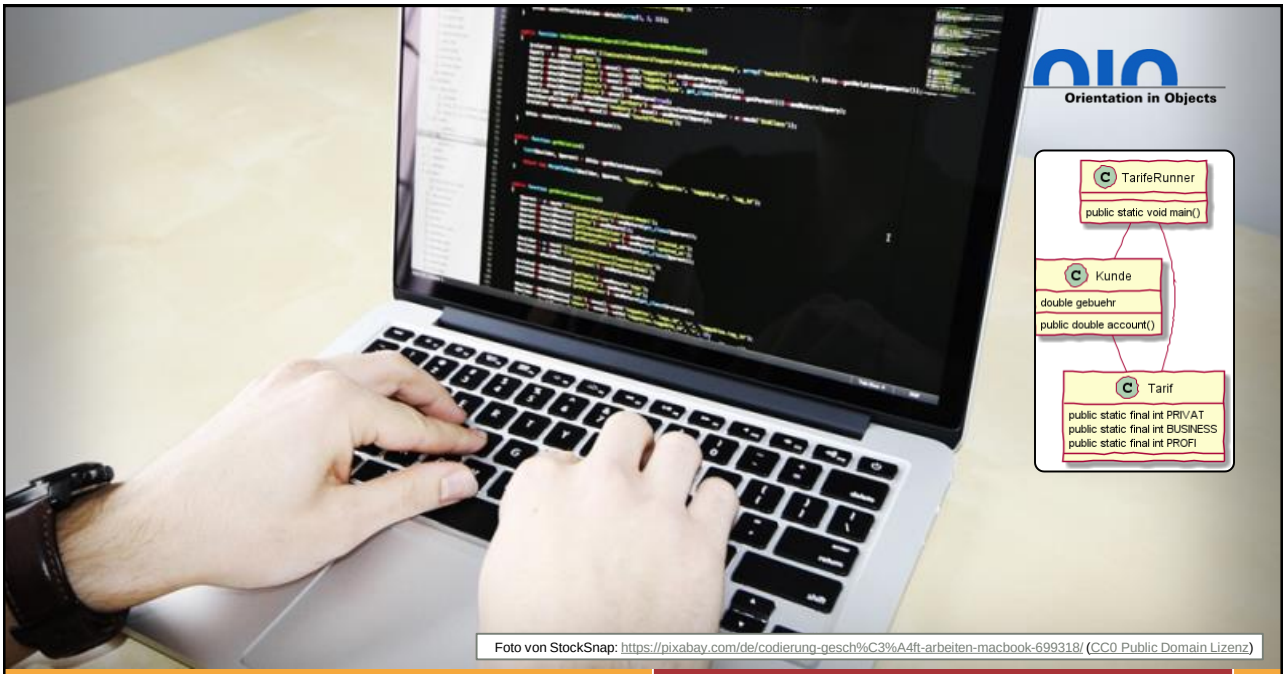
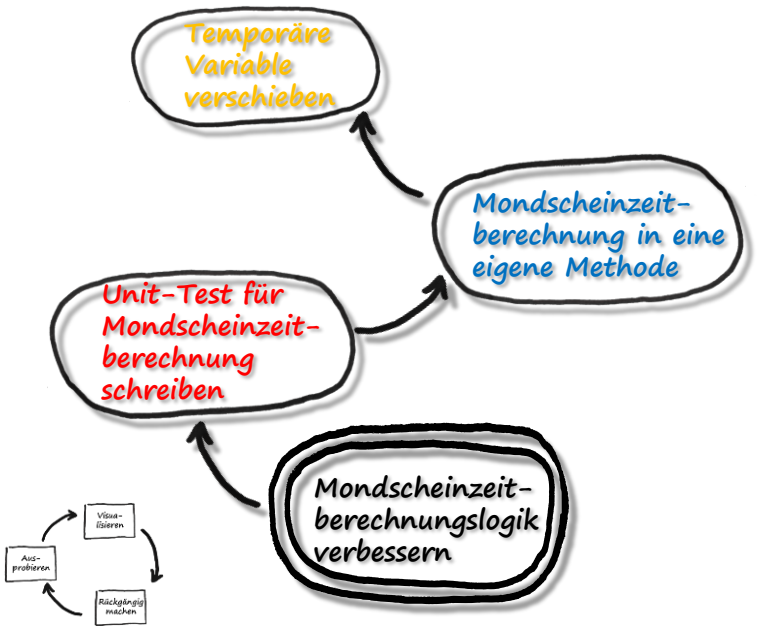


Foto von StockSnap: https://pixabay.com/de/codierung-gesch%C3%A4ft-arbeiten-macbook-699318/ (CC0 Public Domain Lizenz)



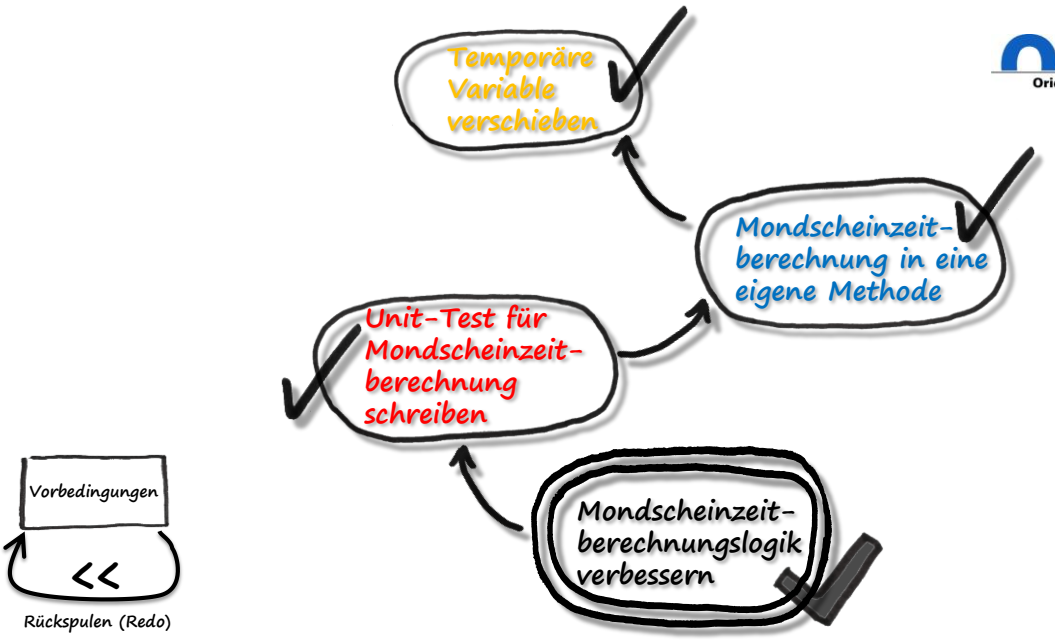
Orientation in Objects

Zurückspulen
Experimentieren

Zurückrollen
Visualisieren
Experimentieren

Zurückrollen
Visualisieren
Experimentieren

Zurückrollen
Visualisieren
Experimentieren
Mikado-Ziel festlegen



Reale Legacy Projekte

Orientation in Objects

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
```

Orientation in Objects

Golden Master
(aka characterization tests)

```java
@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

2

3

```java
public void account(int minuten, int stunde, int minute) {
    System.out.println(String.format("Berechne Gespräch mit
    boolean mondschein = false;
    double preis = 0;
    this.
    // Mon
    if (st
        mo
    // Ges
```

gebuehr : double - Kunde
tarif : Tarif - Kunde
account(int minuten, int stunde, int minute) : void - Kunde
berechnePreis(int minuten, double d) : double - Kunde

retest

recheck

```java
public static void main(String... args) throws Except
  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);
  }
}
```

4

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

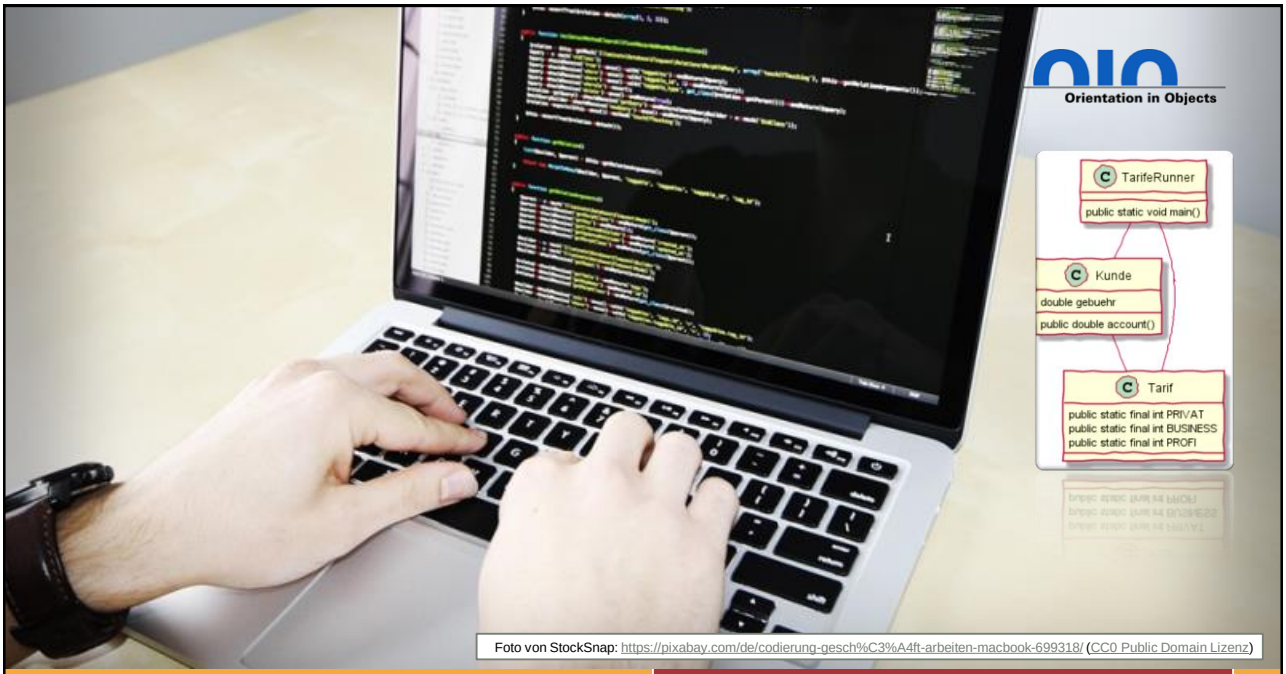
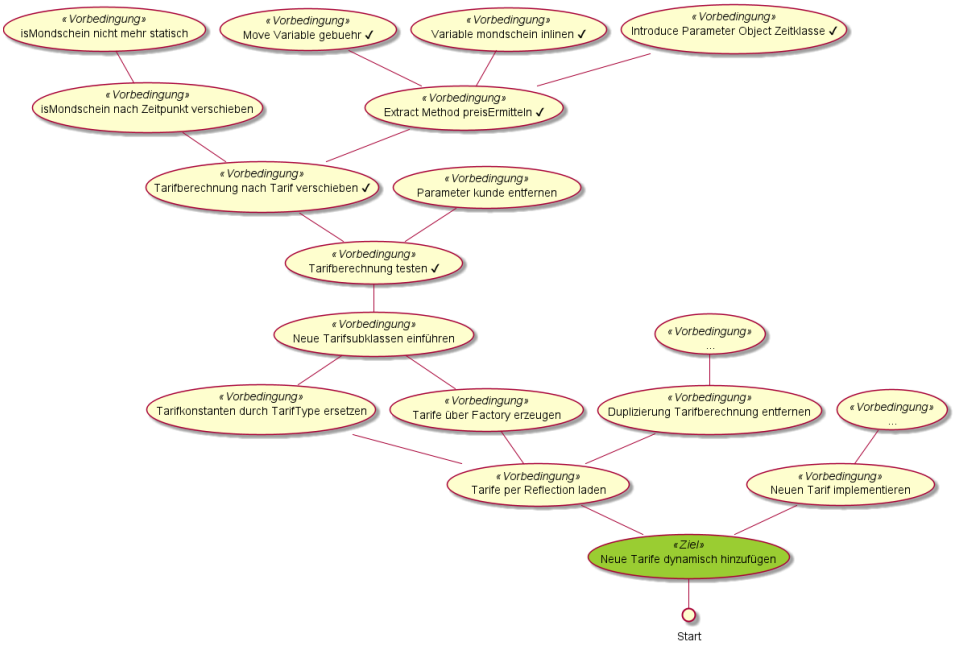


Foto von StockSnap: https://pixabay.com/de/codierung-gesch%C3%A4ft-arbeiten-macbook-699318/ (CC0 Public Domain Lizenz)



Orientation in Objects

Fazit

Orientation in Objects

Vorteile

***Stabiles System** trotz Änderungen.*

*Bessere **Kommunikation** und **Zusammenarbeit**.*

***Leichtgewichtig** und **fokussiert**.*

Orientation in Objects

Wann nutzen?

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

1 *Architektur im Betrieb verbessern*

Orientation in Objects

*In **kleinen Schritten** verbessern.*

*Im **gleichen Branch** neue Features **kontinuierlich ausliefern.***

2 Brownfield Entwicklung

Orientation in Objects

Häufige Situation.

Existierende Anwendungen verbessern.

3 Refactoring Projekt

Orientation in Objects

***Langdauernde** Verbesserung.*

Eigener Branch.

Orientation in Objects

Angst vorm Revert:

Reverting** scheint wie **Wegwerfen.

*Aber Mikado-Graph **enthält Infos.***

***Wissensaufbau/-transfer** über*
System, Domäne, Technologie.

Angst vorm Revert: Änderungen doch aufheben

Orientation in Objects

Stash (Git)

Patch-File (SVN, …)

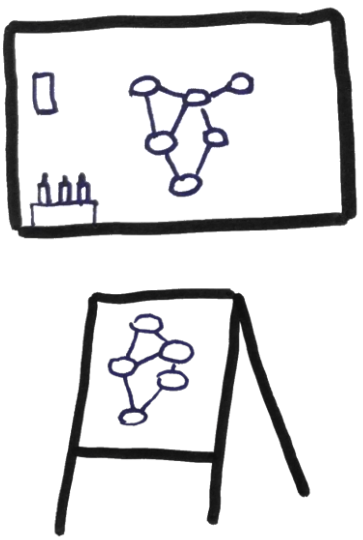
Orientation in Objects

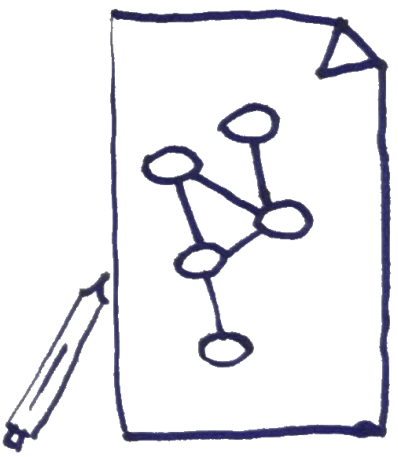
Arbeitsmittel:

1. ***Whiteboard/Flipchart/Papier***
2. ***Mindmap***
3. ***Visio, yEd***

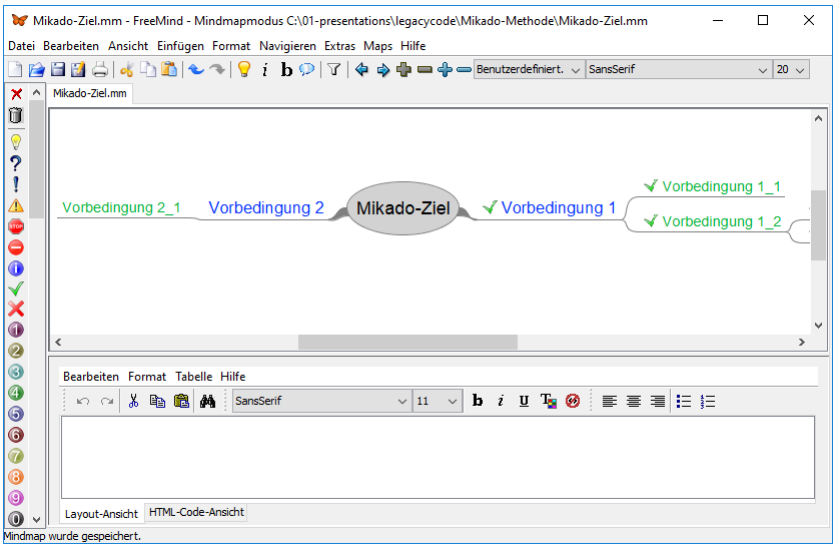
1

Orientation in Objects

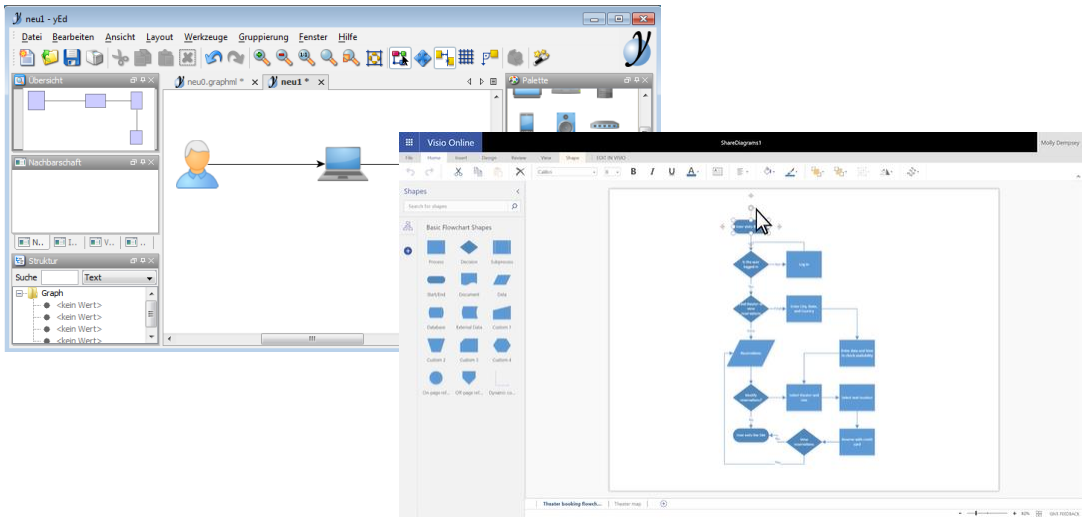


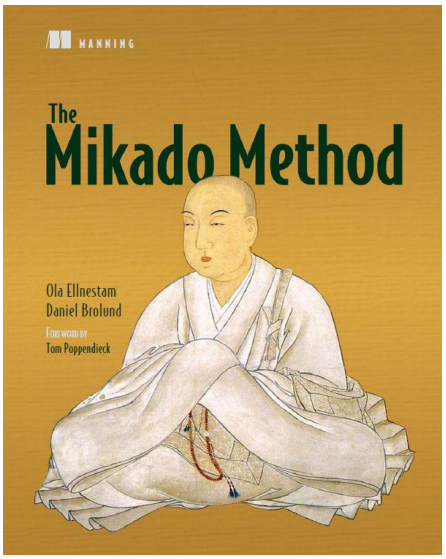


2


Mikado-Ziel.mm - FreeMind - Mindmapmodus C:\01-presentations\legacycode\Mikado-Methode\Mikado-Ziel.mm
Datei Bearbeiten Ansicht Einfügen Format Navigieren Extras Maps Hilfe
Benutzerdefiniert.
SansSerif
20
Mikado-Ziel.mm
Vorbedingung 2_1
Vorbedingung 2
Mikado-Ziel
Vorbedingung 1
Vorbedingung 1_1
Vorbedingung 1_2
Bearbeiten Format Tabelle Hilfe
SansSerif
11
Layout-Ansicht
HTML-Code-Ansicht
Mindmap wurde gespeichert.

3


neu1 - yEd
Datei Bearbeiten Ansicht Layout Werkzeuge Gruppierung Fenster Hilfe
Übersicht
Nachbarschaft
Struktur
Graph
Palette
Visio Online
Shapes
Basic Flowchart Shapes



Vielen Dank für Ihre Aufmerksamkeit!

Orientation in Objects GmbH

Weinheimer Str. 68
68309 Mannheim

www.oio.de
info@oio.de

andrena
OBJECTS

Bitte geben Sie uns jetzt Ihr Feedback!

Legacy Code im Griff dank Mikado Methode

Falk Sippach