



# **CCD Roter Dan**



**I/III**

“How long will it take to master aikido?”  
a prospective student asks.

“How long do you expect to live?”  
is the only respectable response.

- Don't Repeat Yourself
- Keep It Simple Stupid
- Vorsicht vor Optimierungen
- Favour Composition over Inheritance
- Die Pfadfinderregel beachten
- Root Cause Analysis
- Versionskontrollsystem
- Einfache Refaktorisierung
- Täglich reflektieren



- Prinzip
  - Don't Repeat Yourself
  - Keep It Simple Stupid
  - Vorsicht vor Optimierungen
  - Favour Composition over Inheritance
- Praktik
  - Die Pfadfinderregel beachten
  - Root Cause Analysis
  - Versionskontrollsystem
  - Einfache Refaktorisierung
  - Täglich reflektieren



## **Prinzip**

*allgemeingültige Regel, Grundlage,  
auf der etwas aufgebaut ist;  
Grundregel; Grundsatz*

## **Praktik**

*bestimmte Art der Ausübung, Handhabung;  
Verfahrensweise*

- Prinzip
  - **Don't Repeat Yourself**
  - Keep It Simple Stupid (2)
  - Vorsicht vor Optimierungen (2)
  - Favour Composition over Inheritance (3)
- Praktik
  - Die Pfadfinderregel beachten (2)
  - Root Cause Analysis (3)
  - Versionskontrollsystem
  - Einfache Refaktorisierung
  - Täglich reflektieren



# Don't Repeat Yourself (DRY)

## Warum?

*Jede Doppelung von Code oder auch nur Handgriffen leistet Inkonsistenzen und Fehlern Vorschub.*



# Don't Repeat Yourself (DRY)



greek & poke



STACKOVERFLOW

# Don't Repeat Yourself (DRY)

- passiert gerne wenn es schnell gehen muss
- sei dir bewusst wenn du Code/Artefakte wiederholst
- erkenne Wiederholungen
- bereinige durch Refaktorisierungen
  - (wenn keine anderen Prinzipien oder Beschränkungen dagegen sprechen)

# Don't Repeat Yourself (DRY)

**Beispiel:**

`LogicProcessorIF.replaceComponent(String, String)`

**3 Implementierungen**

## LogicComponentProcessor:

@Override

```
public AbsDOPEDocComponent[] replaceComponent(final String oldDomainKey,  
                                              final String newDomainKey)  
                                              throws DOPEException {  
  
    final AbsDOPEDocComponent newElement =  
        this.compSession.getObjectAcquirer().acquire(newDomainKey);  
    DOPERulesUtil.setOrigin(newElement, DocElementOrigin.LOGIC, true);  
    if (this.docStencil != null) {  
        final AbsDOPEDocComponent replaced = this.docStencil.replaceElementBy(oldDomainKey,  
            newElement);  
        if (replaced != null) {  
            final AbsDOPEDocComponent[] result = new AbsDOPEDocComponent[2];  
            result[0] = replaced;  
            result[1] = newElement;  
            return result;  
        }  
    }  
    return null;  
}
```

## DOPERulesForTRW:

@Override

```
public AbsDOPEDocComponent[] replaceComponent(final String oldDomainKey,
                                              final String newDomainKey)
                                              throws DOPEException {

    final AbsDOPEDocComponent newElement =
        this.compSession.getObjectAcquirer().acquire(newDomainKey);
    DOPERulesUtil.setOrigin(newElement, DocElementOrigin.LOGIC, true);
    if (this.masterStencil != null) {
        if (this.masterStencilKey.equals(oldDomainKey)) {
            final DOPEResourceMessage msg = DOPEResourceBundle.getInstance()
                .getMessage(DOPEKernelLocaleIF.RESOURCE_NAME,
                    DOPEKernelLocaleIF.REPLACE_MAIN_STENCIL_NOT_ALLOWED);
            final String[] params = new String[] {oldDomainKey, newDomainKey};
            if (msg.getSeverity().equals(DOPEStatusValuesIF.STATUSSEVERITY_ERROR_S)) {
                throw new DOPEKernelException(DOPEKernelLocaleIF.REPLACE_MAIN_STENCIL_NOT_ALLOWED,
                    params);
            }
        }
        final AbsDOPEDocComponent replaced = this.masterStencil.replaceElementBy(oldDomainKey,
            newElement);
        if (replaced != null) {
            final AbsDOPEDocComponent[] result = new AbsDOPEDocComponent[2];
            result[0] = replaced;
            result[1] = newElement;
            return result;
        }
    }
    return null;
}
```

## SimpleTrwProcessor:

@Override

```
public AbsDOPEDocComponent[] replaceComponent(final String oldDomainKey,
                                              final String newDomainKey)
                                              throws DOPEException {
    if (this.compSession == null || this.masterStencil == null) {
        return null;
    }
    final AbsDOPEDocComponent newElement =
        this.compSession.getObjectAcquirer().acquire(newDomainKey);
    DOPERulesUtil.setOrigin(newElement, DocElementOrigin.LOGIC, true);
    if (this.masterStencil.getDomainKey().equals(oldDomainKey)) {
        final DOPEResourceMessage msg = DOPEResourceBundle.getInstance()
            .getMessage(DOPEKernelLocaleIF.RESOURCE_NAME,
                DOPEKernelLocaleIF.REPLACE_MAIN_STENCIL_NOT_ALLOWED);
        final String[] params = new String[] {oldDomainKey, newDomainKey};
        if (msg.getSeverity().equals(DOPEStatusValuesIF.STATUSSEVERITY_ERROR_S)) {
            throw new DOPEKernelException(DOPEKernelLocaleIF.REPLACE_MAIN_STENCIL_NOT_ALLOWED,
                params);
        }
    }
    final AbsDOPEDocComponent replaced =
        this.masterStencil.replaceElementBy(oldDomainKey, newElement);
    if (replaced != null) {
        final AbsDOPEDocComponent[] result = new AbsDOPEDocComponent[2];
        result[0] = replaced;
        result[1] = newElement;
        return result;
    }
}
```

# Can you see it?

```
try {
    executeTxmComponent(execute == null ? master : execute);
} catch (final DOPEReplayException ex) {
    finishDebugging(txmObj.getDomainKey());
    resetDebugger();
    throw ex;
} catch (final DOPEKernelException ex) {
    LogManager.acquireLM().dumpStackTrace(this, VER, MN, IRecordType.TYPE_ERROR_EXC, ex);
    if (this.rules != null) {
        this.rules.setTrwStatus(new DOPELocaleStatus(ex.getMessage()));
    }
    finishDebugging(txmObj.getDomainKey());
    throw ex;
} catch (final ParserException ex) {
    LogManager.acquireLM().dumpStackTrace(this, VER, MN, IRecordType.TYPE_ERROR_EXC, ex);
    if (this.rules != null) {
        this.rules.setTrwStatus(new DOPELocaleStatus(ex.getMessage()));
    }
    handleParserException(ex);
} catch (final DOPEException ex) {
    LogManager.acquireLM().dumpStackTrace(this, VER, MN, IRecordType.TYPE_ERROR_EXC, ex);
    if (this.rules != null) {
        this.rules.setTrwStatus(new DOPELocaleStatus(ex.getMessage()));
    }
    finishDebugging(txmObj.getDomainKey());
    throw new DOPEException(ex.getMessage(), ex);
}
```

# Can you see it?

```
try {
    executeTxmComponent(execute == null ? master : execute);
} catch (final DOPEReplayException ex) {
    finishDebugging(txmObj.getDomainKey());
    resetDebugger();
    throw ex;
} catch (final DOPEKernelException ex) {
    LogManager.acquireLM().dumpStackTrace(this, VER, MN, IRecordType.TYPE_ERROR_EXC, ex);
    if (this.rules != null) {
        this.rules.setTrwStatus(new DOPELocaleStatus(ex.getMessage()));
    }
    finishDebugging(txmObj.getDomainKey());
    throw ex;
} catch (final ParserException ex) {
    LogManager.acquireLM().dumpStackTrace(this, VER, MN, IRecordType.TYPE_ERROR_EXC, ex);
    if (this.rules != null) {
        this.rules.setTrwStatus(new DOPELocaleStatus(ex.getMessage()));
    }
    handleParserException(ex);
} catch (final DOPEException ex) {
    LogManager.acquireLM().dumpStackTrace(this, VER, MN, IRecordType.TYPE_ERROR_EXC, ex);
    if (this.rules != null) {
        this.rules.setTrwStatus(new DOPELocaleStatus(ex.getMessage()));
    }
    finishDebugging(txmObj.getDomainKey());
    throw new DOPEException(ex.getMessage(), ex);
}
```



# Can you see it?

```
try {
    executeTxmComponent(execute == null ? master : execute);
} catch (final DOPEReplayException ex) {
    finishDebugging(txmObj.getDomainKey());
    resetDebugger();
    throw ex;
} catch (final DOPEKernelException ex) {
    LogManager.acquireLM().dumpStackTrace(this, VER, MN, IRecordType.TYPE_ERROR_EXC, ex);
    if (this.rules != null) {
        this.rules.setTrwStatus(new DOPELocaleStatus(ex.getMessage()));
    }
    finishDebugging(txmObj.getDomainKey());
    throw ex;
} catch (final ParserException ex) {
    LogManager.acquireLM().dumpStackTrace(this, VER, MN, IRecordType.TYPE_ERROR_EXC, ex);
    if (this.rules != null) {
        this.rules.setTrwStatus(new DOPELocaleStatus(ex.getMessage()));
    }
    handleParserException(ex);
} catch (final DOPEException ex) {
    LogManager.acquireLM().dumpStackTrace(this, VER, MN, IRecordType.TYPE_ERROR_EXC, ex);
    if (this.rules != null) {
        this.rules.setTrwStatus(new DOPELocaleStatus(ex.getMessage()));
    }
    finishDebugging(txmObj.getDomainKey());
    throw new DOPEException(ex.getMessage(), ex);
}
```

# Versionskontrollsystem

## Warum?

*Angst vor Beschädigung eines "running system" lähmt die Softwareentwicklung*

*Mit einer Versionsverwaltung ist solche Angst unbegründet. Die Entwicklung kann schnell und mutig voranschreiten.*

# Versionskontrollsystem

*Ein Versionskontrollsystem nimmt die Angst,  
etwas falsch und damit kaputt zu machen.*

*Keine Angst einen erreichten Stand zu zerstören.*

*Nichts geht verloren.*

# Einfache Refaktorisierung

## Warum?

*Code verbessern ist leichter, wenn man typische Verbesserungshandgriffe kennt.*

*Ihre Anwendungsszenarien machen sensibel für Schwachpunkte im eigenen Code.*

*Als anerkannte Muster stärken sie den Mut, sie anzuwenden.*

# Einfache Refaktorisierung

- *Pflichtlektüre Fowler*
- *Methode extrahieren*
- *Umbenennen*

# Täglich reflektieren

## Warum?

*Keine Verbesserung, kein Fortschritt,  
kein Lernen ohne Reflexion.*

*Aber nur, wenn Reflexion auch eingeplant wird,  
findet sie unter dem Druck des Tagesgeschäftes  
auch statt.*

# Täglich reflektieren

- Pfadfinderregel des CCDs auf sich selbst angewandt
- ohne ein "Kontrollsystem" braucht es:
  - kleinschrittige Planung (max. 1 Tag/Task)
  - Reflexion nach jedem Schritt
- am Abend jeden Tages eine Bilanz ziehen
  - Habe ich alle meine Aufgaben erledigt?
  - Wie habe ich meine Aufgaben erledigt?
  - Alle nach CCD-Grad relevanten Aspekte des Wertesystems berücksichtigt?

# Täglich reflektieren

- optionale haptische Unterstützung des Lernens
  - linker/rechter Arm
- 21 Tage lang nicht gewechselt
  - nächster Grad





# Üben, üben, üben...



- CodeKata
  - Supermarket Pricing
  - Karate Chop
  - How Big, How Fast?
  - ...

Eine Kata (jap. 形 oder 型) im Karate ist – wie in anderen japanischen Kampfsportarten auch – eine Übungsform, die aus stilisierten Kämpfen besteht, welche jedoch im Karate meist gegen imaginäre Gegner geführt werden.

[Wikipedia]

„Mastering is more than just knowing.  
It is knowing in a way that lightens  
your load.“

<http://www.clean-code-developer.de/>