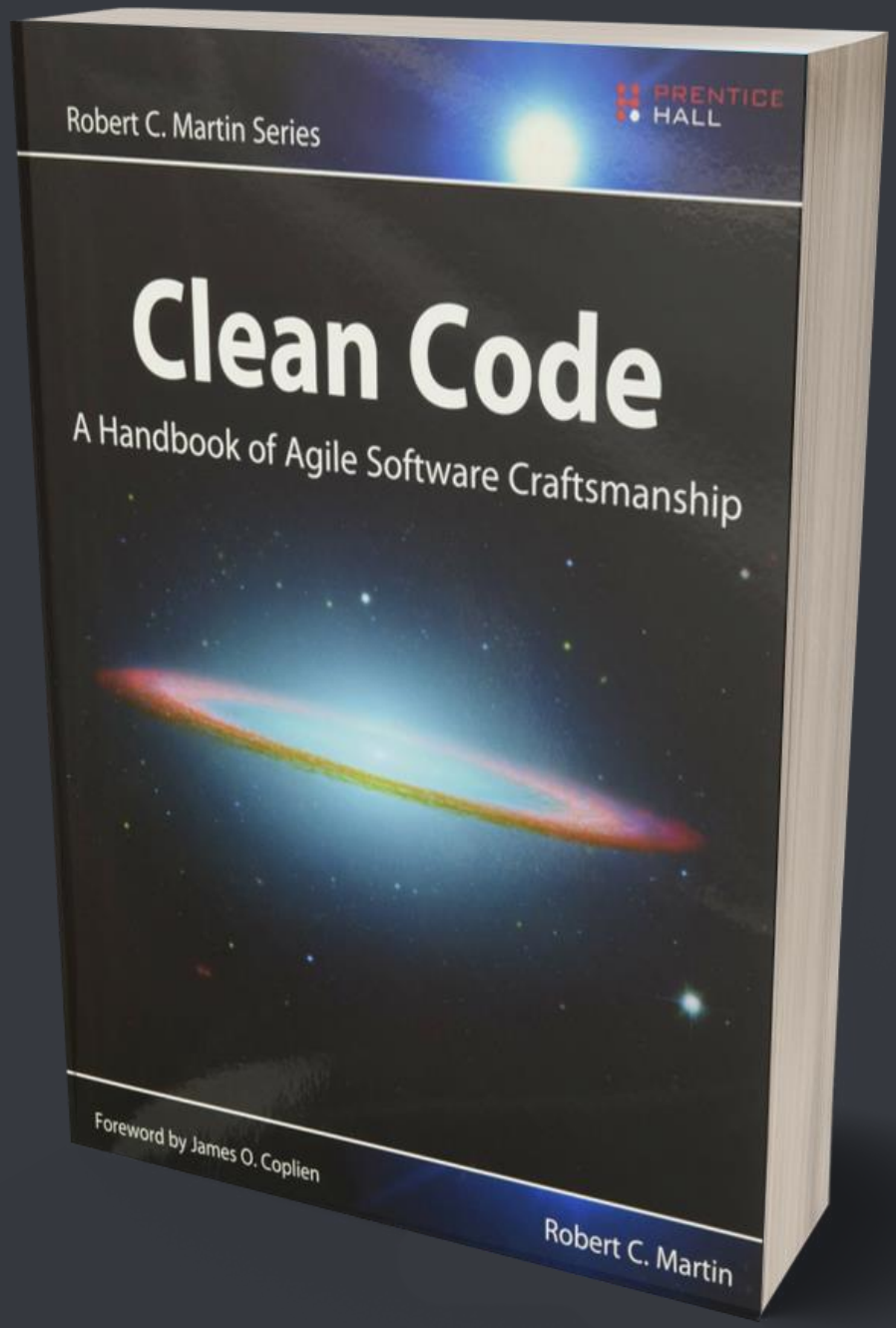
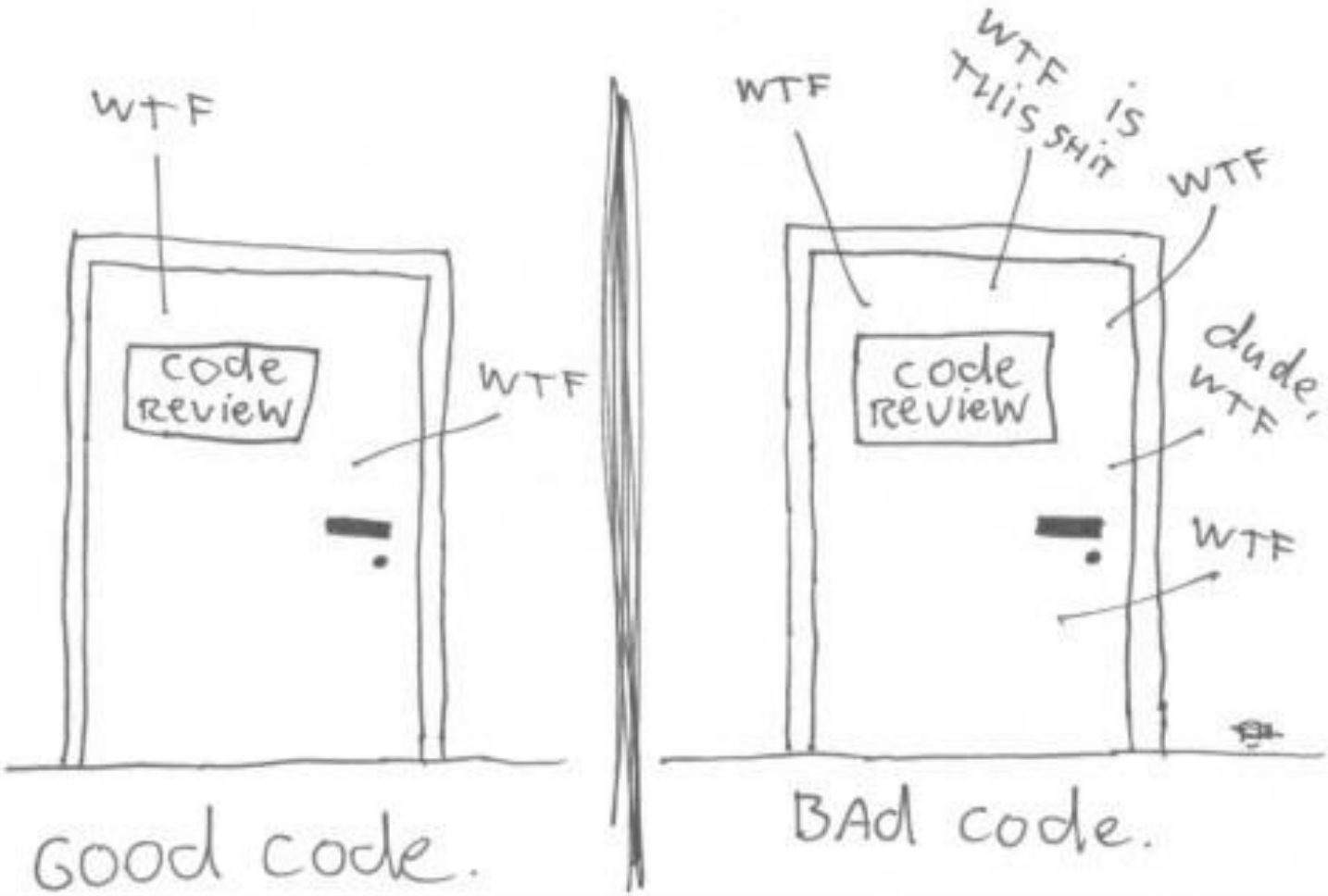

Robert C. Martin Series
PRENTICE HALL
Clean Code
A Handbook of Agile Software Craftsmanship
Foreword by James O. Coplien
Robert C. Martin

The ONLY VALID MEASUREMENT OF CODE QUALITY: WTFs/MINUTE



Programmieren ist eine

soziale Aktivität.

Handle auch so ☺

Denke in “-lties”

- maintainability
- extensibility
- testability
- reusability
- …

Sieben ausgewählte Clean Code Rules

1. Meaningful Names
2. Comments Only Where Necessary
3. Keep Your Methods Small
4. No Side Effect in a Method
5. Only One Level of Abstraction
6. Stepdown Rule
7. Proper Error Handling (Exceptions)

Bild: Patrick, https://www.flickr.com/photos/giantginkgo/37740313

1) Meaningful Names

Intention-Revealing Names

```
int d; // elapsed time in days
```

```
int elapsedTimeInDays;
```

Avoid Disinformation

```
ISet accountList =
        new HashSet<Account>();

ISet accountGroup =
        new HashSet<Account>();
```

Meaningful Distinctions

```
public static void copyChars(char[] a1, char[] a2) {
    for (int i = 0; i < a1.Length; i++) {
        a2[i] = a1[i];
    }
}

public static void copyChars(char[] source, char[] destination) {
    for (int i = 0; i < source.Length; i++) {
        destination[i] = source[i];
    }
}
```

Pronounceable Names

```
class DtaRcrd102 {
    private DateTime genymdhms;
    private DateTime modymdhms;
    private const string pszqint = "102";
    /* ... */
};

class Customer {
    private Date generationTimestamp;
    private Date modificationTimestamp;
    private const string recordId = "102";
    /* ... */
};
```

Searchable Names

```
int i;
float res;
string st;
```

- Variablen mit 1-3 Buchstaben nur im lokalen Kontext!

Global vs. Local

- Deklaration nahe bei Verwendung
- Lieber lokal als global

Naming Rules

- Class = Hauptwort

```
class Customer {
    //...
};
```

- Method = Verb

```
if ( paymentMethod.isValid() ) {
    customer.ActivatePaymentMethod( paymentMethod );
}
```

One Word per Concept (and Stick To It)

```
FetchXXX(...);
//...
RetrieveYYY(...);
//...
GetZZZ(...);
```

```
FetchXXX(...);
//...
FetchYYY(...);
//...
FetchZZZ(...);
```

2) Comments only where necessary

Bild: Alberto Ortiz,
https://www.flickr.com/photos/citizen_poeta/1446906402

“Indeed, comments are, at best, a necessary evil.”

“Comments are always failures.”

(Bob Martin, Clean Code)

Do’s

- Rechtliche Anmerkungen (Copyright, Lizenz, etc.)
- Erklärung einer Absicht
  - Aus welchem Grund hast du als Entwickler diese Entscheidung getroffen?
- Klarstellung
- Warnung vor Konsequenzen einer Änderung
- TODOs
- Dokumentation von Methodenköpfen

Don’ts

- Erklär deinen Code nicht
  - Wir sind Entwickler, wir können Code lesen.
- Doppelt genäht hält NICHT besser
  - Redundante Information altert schlecht – wird nämlich niemals aktualisiert ☹
- Unpräzise gesprochene Sprache
  - Deshalb nutzen wir Programmiersprachen!

```
// If the application is complete, the attached documents
// can be sent for checking. Otherwise, this operation is
// not (yet) allowed.
if ( State() == DOCUMENTS_UNCHECKED ) {
    if ( application.IsComplete() ) {
        application.SendDocumentsForChecking();
    }
    else {
        throw new InvalidOperationException (
            “Documents can be sent for checking only if " +
            “the application is complete.” );
    }
}
```


3) Keep
your
methods
small


1 Methode =
300 Zeilen

4) No side effect in a method

Bild: Marcel Oosterwijk,
https://www.flickr.com/photos/wackelijmrooster/3177929150

- **Side effect (Seiteneffekt) =**
- Deine Methode “verspricht” das eine.
- Aber sie macht auch noch “versteckte” andere Dinge.

Inputs & Outputs

- Valide **Inputs** sind
  - die Parameter der Methode
  - Properties/Fields des eigenen Objekts (“this”)
- Valide **Outputs** sind
  - Properties/Fields des eigenen Objekts (“this”)
  - der Rückgabewert

```
public bool IsValidPassword(
                                String passwordString ) {
    bool syntaxOk = false;
    //… test if password matches Syntax …
    if ( syntaxOk ) {
        Session.initialize();
        return true;
    }
    return false;
}
```

```
public bool IsValidPassword(
                                String passwordString ) {
    bool syntaxOk = false;
    //… test if password matches Syntax …
    if ( syntaxOk ) {
        Session.initialize();
        return true;
    }
    return false;
}
```

Command Query Separation

- Name impliziert Command oder Query
  - Mere queries should not change anything!

```
public List<StatusInfo> GetStatus() {
    //… collect status infos …
    SetAccessTime( … );
    SetRequestedStatusInfo( … );
    SetDefaultRequest( … );
    return …; // the requested infos
}
```

Command Query Separation

- Name impliziert Command oder Query
  - Mere queries should not change anything!

```
public List<StatusInfo> GetStatus() {
    //… collect status infos …
    SetAccessTime( … );
    SetRequestedStatusInfo( … );
    SetDefaultRequest( … );
    return …; // the requested infos
}
```

5) Only one level of abstraction

```
public class ProductCatalogue {
    public string AsMarkdown() {
        string metaInfos = GetMetaInfos().AsMarkdown();
        string productPages = “”;
        for (product : productList) {
                productPages += product.AsMarkdown();
        }
        String footer =
                “© 2021 XYZ Ltd. \n\n All rights reserved!\n” +
                “**No authorized copying!**\n” +
                “Violations please report to ” +
                getResponsibleManager().GetEmail() + “\n”;
       return (metaInfos + productPages + footer);
    }
}
```

```
public class ProductCatalogue {
    public string AsMarkdown() {
        string metaInfos = GetMetaInfos().AsMarkdown();
        string productPages = “”;
        for (product : productList) {
                productPages += product.AsMarkdown();
        }
        String footer =
                “© 2021 XYZ Ltd. \n\n All rights reserved!\n” +
                “**No authorized copying!**\n” +
                “Violations please report to ” +
                getResponsibleManager().GetEmail() + “\n”;
      return (metaInfos + productPages + footer);
    }
}
```

```
public class ProductCatalogue {
    public string AsMarkdown() {
        string metaInfos = GetMetaInfos().AsMarkdown();
        string productPages = “”;
        for (product : productList) {
                productPages += product.AsMarkdown();
        }
        String footer = getCopyRightFooter().asMarkdown();
        return (metaInfos + productPages + footer);
    }
}
```

6) Stepdown Rule

"Wir wollen, dass sich der Code wie eine Erzählung von oben nach unten liest.

Wir wollen, dass auf jede Funktion die Funktionen der nächsten Abstraktions-ebene folgen.

So können wir das Programm lesen, indem wir eine Abstraktionsebene nach der anderen absteigen, während wir die Liste der Funktionen abwärts lesen."

```
private void cook() {
    fryingPan.mixContents();
    fryingPan.add(salt.getABit());
    fryingPan.mixContents();
}

public void makeBreakfast() {
    addEggs();
    cook();
    wife.give(fryingPan.getContents(20, PERCENT));
    self.give(fryingPan.getContents(80, PERCENT));
}

private void addEggs() {
    fridge
            .getEggs()
            .forEach(egg -> fryingPan.add(egg.open()));
}
```

Beispiel nach:
https://dzone.com/articles/the-stepdown-rule

```
public void makeBreakfast() {
    addEggs();
    cook();
    serve();
}

private void addEggs() {
    fridge
            .getEggs()
            .forEach(egg -> fryingPan.add(egg.open()));
}

private void cook() {
    fryingPan.mixContents();
    fryingPan.add(salt.getABit());
    fryingPan.mixContents();
}

private void serve() {
    wife.give(fryingPan.getContents(20, PERCENT));
    self.give(fryingPan.getContents(80, PERCENT));
}
```


7) Proper
Error
Handling
(Exceptions)

“Prefer exceptions to returning error codes.”

- Exceptions sind Störungen im normalen Programmablauf
  - (so wie das Ausnahmeszenario eines Use Cases)
- Exception = kontrollierter Umgang mit Störungen

www.archi-lab.io