
# Validating the Object Calisthenics

## Evaluation and Prototypical Implementation of Tool Support

## Student Research Paper

for the certification examinations for the

Bachelor of Science

Degree Course Applied Computer Science

Baden-Wuerttemberg Cooperative State University Karlsruhe

by

Fabian Schwarz-Fritz

| | |
|---|---|
| Publication date: | November 21, 2013 |
| Time required for processing: | 12 Weeks |
| Matriculation number: | 212024979 |
| Course: | TINF11B2 |
| Vocational training company: | SAP AG, Walldorf |
| Reviewer: | Daniel Lindner |
| Reviewer's company: | Softwareschneiderei GmbH, Karlsruhe |

# Eidesstattliche Erklärung

Ich versichere hiermit, dass ich meine Bachelorarbeit mit dem Thema

> *Validating the Object Calisthenics - Evaluation and Prototypical Implementation of Tool Support*

selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Die Arbeit wurde bisher keiner anderen Prüfungsbehörde vorgelegt und auch nicht veröffentlicht.

Mir ist bekannt, dass ich meine Diplomarbeit zusammen mit dieser Erklärung fristgemäß nach Vergabe des Themas in dreifacher Ausfertigung und gebunden im Sekretariat meines Studiengangs an der DHBW Karlsruhe abzugeben habe. Als Abgabetermin giltbei postalischer Übersendung der Eingangsstempel der DHBW, also nicht der Poststempel oder der Zeitpunkt eines Einwurfs in einen Briefkasten der DHBW.

Karlsruhe, den November 21, 2013

---

Fabian Schwarz-Fritz

---

## Sperrvermerk - TODO ??? INFORMIEREN. Gehoert das der SAP AG

Die Ergebnisse der Arbeit stehen ausschließlich dem auf dem Deckblatt aufgeführten Ausbildungsbetrieb zur Verfügung.

# Abstract

Hier bitte den Abstract Ihrer Arbeit eintragen. Der Abstract sollte nicht länger als eine halbe Seite sein. Bitte klären Sie mit Ihrem Studiengangsleiter ab, ob der Abstract in englischer oder deutscher Sprache (oder möglicherweise sogar in beiden Sprachen) verfasst werden soll.

# Contents

# Abbreviations

| | |
|---|---|
| DHBW | Duale Hochschule Baden-Württemberg |
| OSS | Open Source Software |
| Sem | Semester |

# List of Figures

# List of Tables

# 1 Introduction

Jeff Bay's "Object Calisthenics" [Bay, 2008] are nine rules that train the software developer to write better object oriented code. He created the concrete rules out of general software principals and patterns. These rules shall be applied in a short excercise, usually about two to four hours. With these concrete rules the trainee doing the excercise can improve his software development skills, which is helping him when applying general software principals and patterns to real world software projects. The excercise and the reasons for it are described in this chapter, in section [**?**].

This paper is divided into three chapters.

Chapter 2 describes Jeff Bay's rule and why he was led to the nine rules. It poses Jeff Bay's reasons for today's problems of software development. The chapter describes every rule step by step. Within the step by step description for ever rule, an source code example is given. A bad example firstly exemplifies how the code looks like without the rule. A good example, validating the rule, then shows the advantages of the resulting source code when applying the rule. These examples are described shortly. After describing the examples, it researches concepts behind the rule. It describes the problems that would occur without the rule. After explaining the problems behind the rule and Jeff Bay's rule, a more detailed analysis ensues. In this detailed analysis, patterns and principals are described and ideas and concepts are examined. Section 2.2 describes these principles detailed for every rule, step by step. In the end, the section 2.3 furthermore discusses the rules shortly.

Chapter 3 investigates possible tool support to validate the Object Calisthenics. First, the section 3.1 describes general advantages of tool support. The section 3.3 goes through every rule presents the possibilities to validate the given rule. Advantages and disadvantages of the implementation are explained and a short conclusion of every rule is drawn. Section 3.4 then summarizes the results of the evaluation of rule validation. Lastly, section 3.5 further ideas of rule validation and links for further work.

The last chapter 4 describes a prototypical implementation done during this research. The prototype shows the practicability of the evaluation results.

## 1.1 Exercizing Better Object Oriented Programming Skills: a Concept by Jeff Bay

Jeff Bay's "Object Calisthenics" [Bay, 2008] is chapter from a book "The Thoughtworks Anthology" [ThoughtWorks inc., 2008]. in the chapter he describes nine rules to that should be applied in trainings.

=== What are the Object Calisthenics in general (nine rules...)

=== What is the paper where the OC are presented?

=== Who is Jeff Bay and what is special about ThoughtWorks inc?

=== What is "the excercise": break habits, etc....? Describe stuff from research phase. Do not describe the rules, but do some references to chapter 2.

=== What is the outcome according to Jeff Bay?

=== Say: The research s done in chapter 2

*// Some first paragraphs to get used to tex writing. This is far form beeing a final version... yada yada yada:*

*The Object Calisthenics are nine programming rules helping to write good object oriented code.*

*The paper capter "The Object Calisthenics" describes the purpose, the outcome and the rules. It was released in 2008 in the paper "The thoughtworks anthology. Essays on Software Technology and Innovation"[ThoughtWorks inc., 2008, p. 70-79]. The whole paper consists of thirteen chapters discussing various topics and ideas on how to improve software development. On of the paper's chapter describes the rules of the Object Calisthenics and their purpose and outcome, shortly.*

*All essays in the paper are written by developers working at the company "Thoughtworks inc". ThoughtWorks is well known for creating, designing and supporting high quality software. The company is to be said to be one of the most future oriented company in terms of technology and software principals. They describe themselves as "[...]a software company and community of passionate inidividuals whose purpose is to revolutionize software design, creation and delivery, while advocating for positive social change[...]"[ToughtWorks inc.]*

*The Object Calisthenics are an exercise to improve the quality of Object Oriented code. Good Object Oriented Code is hard to learn when coming from procedural code. Many developers think in Object Oriented code – but do they really write good Object Oriented Software? That is the question that Jeff Bay poses in his essay. Usually the developer doesn't use these rules in real world project but applies them in short two hour exercises in which he designs and implements minimalistic software with little requirements. This could be a Minesweeper or a TicTacToe game for example. These training challenges should lead the developer to write better code and be more aware of code quality in real world projects.*

*// Hey, don't forget:*

*RESEARCH,l page 22: spen 20 hours, 1000 loc, break habits, etc....*

*With these little training sessions, the Object Calisthenics help to create highly object oriented code in small projects. When applying the rules, the developers automatically fulfill many important software patterns and principals leading to higher code quality than the code would have without the given rules. By training developers to focus the rules they automatically apply various helpful and important software principals and software patterns.*

*// Some more about how the excercise is implemented*

*Add: RESEARCH page 5: the excercise, strict!*

*However the idea is that the developers recoginzes the value of the resulting code and then applies parts of the rules or the principals behind them to larger, real world projects. Improving the quality of software's implementation by little training sessions - that is the basic idea. Jeff Bay, the author of the paper, included this idea also in the name of the rules. The word "Calisthenics" undistinctable describes the approach, the idea and the outcome of the exercise.*

## 1.2 Tool Support to Validate the Object Calisthenics

=== Describe the outcome of a tool validating the rules of the Object Calisthenics.

=== How might it support the developer?

=== How might the tool boost the speed and efficiency of the developer while doing the exercise?

=== Say: Tool support is examined and elucidated in chapter 3.

=== Say: Prototype in the last chapter: 4.

*// Yada, yada, yada:*

*The purpose of the Object Calisthenics was already described. However, in this paper the focus lies on the evaluation and prototypical implementation of tool support, validating the rules of the Object Calisthenics.*

*The outcome of tool support validating the source code written during a Object Calisthenics session is the following. The tool support might shorten the time of the training and furthermore guarantees that the developer sticks to the given rules.*

*- Now what is the outcome of tool support for the Object Calisthenics?*

*Therefore, the next chapter 2 describes the rules Object Calisthenics to understand the software principals and quality metrics behind the Object Calisthenics. The ensuing chapter 3 discusses tool support for every rule.*

# 2 Object Calisthenics by Jeff Bay - Patterns and Principals

=== This chapter describes the patterns and principals behind the Object Calisthenics.
=== General introduction and introduction of the paper. Qualities, research stuff.
=== Describe the structure of the sections
=== Important for this chapter: Use correct quotations when talking about Jeff's ideas and rules. Use correct quotations when referencing to other books that are given in the research material.

## 2.1 Advantages of Object Oriented Programming

=== Describe the advantages of Object Oriented Programming according to Jeff...
===RESEARCH=== poorly written code, procedural code
Describe each very shortly:
no reusability
hard to maintain
overview
structure
no bundle o data and behavior
no modularity
not understandable
maintainability is hard
OO saves us! –> But ... because
Comparision of procedural versus object oriented programming procedural: step by step, seldom informaiton hiding, actions mainulate data. Actions are spread all ove rthe programming
oo: "bundle" with capabilities, hides structures, delegates tasks that are not done by the object itself to other objects, objects model real world behaviour, decoupled and separated in different modules. "what" leads to "how": encapsulation, abstraction, inheritance, polymorphism
advantages of oop: modules, reusable maintainable, simplicity (describe each shortly)
Describe each quality, a bit more detailes, but also short and concise:
cohesion
loose coupling
zero duplication
encapsulation
testability
readability
focus

## 2.2 The Rules and their Background

=== This chapter describes the rules of the Object Calisthenics itself.

=== Reference to introduciton, to make sure the reader know that this is an exercise... The exercise: repeat strict coding standards and stuff from introduction shortly

=== FOR EACH rule/subsection IN rules/subsections: Every rule is described one after the other with the information that is already documented in the commited work of the "research" phase:

- Explain the rule. Use quotes of paper.

- Every chapter gives short explanation, a good and a bad example. These examples are explained shortly.

- Furthermore every chapter describes the software patterns and software principals behind every rule. These are for example design patterns, software principals and best practices.

- Refer to other sources to be able to explain clearly but shortly. Summarize complex patterns instead of explaining every pattern and principal in detail.

- Make sure the principals behind are easy to read: A advanced reader should not be bored by detailed explanation and a beginner reader should be able to understand the main message and idea behind the principal and idea.

- The outcome of every rule is then summarized.

### 2.2.1 Rule 1: "Use One Level of Indentation per Method"

I am able to add code directly in the text

```
1 Listings
```

in the text. Furthermore it is possible refer to a file:

```
1  package ocanalyzer.rules.noelse;
2
3  import java.util.ArrayList;
4  import java.util.List;
5
6  import ocanalyzer.rules.general.ValidationHandler;
7
8  import org.eclipse.jdt.core.dom.ASTVisitor;
9  import org.eclipse.jdt.core.dom.IfStatement;
10 import org.eclipse.jdt.core.dom.Statement;
11
12 /**
13  *
14  * This class is used to visit all if {@link Statement}.
15  *
16  * An if {@link Statement} which does have a corresponding else
17  * {@link Statement} is saved and furthermore it is reported to the
       given
18  * {@link ElseValidationHandler}
19  *
20  * @author Fabian Schwarz-Fritz
21  *
```

```
22  */
23 public class ElseVisitor extends ASTVisitor {
24
25         private List<Statement> elseStatements;
26         private ValidationHandler validationHandler;
27
28         public ElseVisitor(ValidationHandler validatonHandler) {
29                 this.validationHandler = validatonHandler;
30                 elseStatements = new ArrayList<Statement>();
31         }
32
33         @Override
34         public void endVisit(IfStatement ifStatement) {
35                 if (isSingleElse(ifStatement)) {
36                         Statement elseStatement = ifStatement.
                            getElseStatement();
37                         elseStatements.add(elseStatement);
38                         validationHandler.printInfo(elseStatement);
39                 }
40         }
41
42         private boolean isSingleElse(IfStatement ifStatement) {
43                 return ifStatement.getElseStatement() != null;
44         }
45
46         public List<Statement> getElseStatements() {
47                 return elseStatements;
48         }
49
50 }
```

which is cool!

Furthermore like this class `ElseVisitor` and I also like `SomeOtherClass`. `IfThereIsAStupid-LineBreakInTheClassName` - what happens then?

### 2.2.2 Rule 2: "Don't Use the else Keyword"

asdf

### 2.2.3 Rule 3: "Wrap All Primitives and Strings"

asdf

### 2.2.4 Rule 4: "Use Only One Dot per Line"

asdf

### 2.2.5 Rule 5: "Don't Abbreviate"

asdf

### 2.2.6 Rule 6: "Keep All Entities Small"

asdf

### 2.2.7 Rule 7: "Don't Use Any Classes with More Than Two Instance Variables"

asdf

### 2.2.8 Rule 8: "Use First-Class Collections"

asdf

### 2.2.9 Rule 9: "Don't Use Any Getters/Setters/Properties"

asdf

## 2.3 Discussing the Rules

2.3 === This chapter is discussing the rules shortly.
=== Use quotes from Jeff's text: He also gives a short summary of his text in the end

### 2.3.1 Similarities

=== Are there similarities in the rules? Is it possible to categorize the rules in terms of: - Do they have the same intention? - Are they related to the same "big picture" idea (example: encapsulation or abstraction)

Categorize rules: Are there similarities from perspective of principle? ??? Together with next chapter?

### 2.3.2 Precedence

=== Make clear that this is my own estimation and not related to Jeff's text. Prioritize the rules.
=== To be determined: How long is this chapter?
*// Yada yada yada: Own estimation: what's the most important rule? What do I think? What does the author think? Reason with descriptions and examples given*

### 2.3.3 Conclusion and Outcome

=== Give a short and precise conclusion of the Object Calisthenic: Purpose, exercise, stuff to learn, skill improvement.
=== Also Summarize what most of the rules are about:
RESERACH page 21:
- behaviour and operation oriented
- LoD and loose coupling
- talk to friends

- talk the protocol specified by the object's operation
- Next page: conclusion, no else ,naming,
all in all: Duplication of code and idea ==> "Simple and elegant Abstractions"

# 3 Tool Support to Validate the Object Calisthenics - Evaluation

## 3.1 Advantages of Tool Support

=== What is a tool? Why do tools help? What makes tools strong? Why do tools matter for developers? Possible outcome of tool support for the OC's? Already described in the Introduction (chapter 1). Describe this more in detail here if necessary.

## 3.2 Working Environment

=== Describe AST generally. Say that Eclipse provides types representing the parts of code syntax. This is seen as given.

=== Refer to other references explaining AST.

=== Describe shortly how it is possible to do an AST validation with eclpise.

=== Say that: "Eclipse" terms for AST nodes are quite similar to general terms for java's nodes in ASTs. In this report the standard terms are used. These are not further described in this paper.

–> Example: Describe that "MethodDeclaration" consists of different other nodes representing the declaration of a method. Describe the structure of the child nodes of the node as far necessary. Do not embark on a discussion about what exactly is allowed as method declaration.

=== Therefore: Give reference for questions about "parameter", "type", "class", "expression" or "statement". Say that the validation in the next section is exactly implemented as described. One validation implementation example is explained exemplary in the Prototype chapter.

## 3.3 Evaluation of Rule Validation

=== Say that the prioritization of the rules (in terms of the rule validation) and the "ranking" is given in the end. The sections of this chapter are "rule specific", even if the next subsections refer to each other.

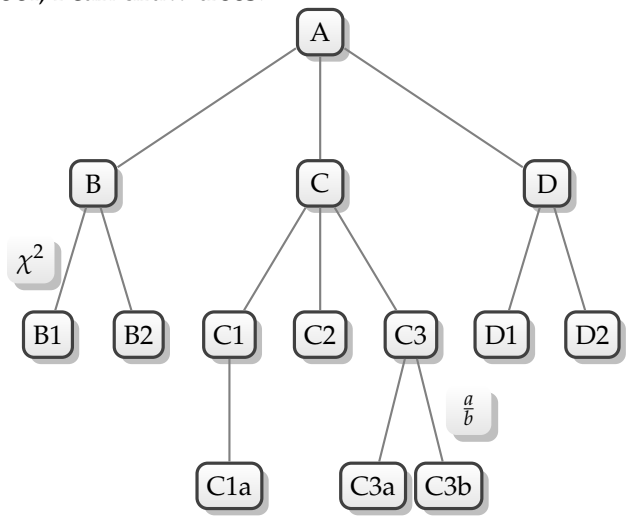
=== FOREACH Rule/Subsection IN Rules/Subsections:

- Similarities found out in description may be similar in this validation? Categorize the rules in groups to form a validation perspective. (E.g.: Validation of rule y is very similar to the validation of rule x that was already explained.)
- Use examples given in description chapter to describe the typical structure of the rule.
- Explain "the positive case": What is the positive structure, satisfying the rule. Where does a rule violation occur when the structure is not fulfilled. (Example: Positive case: maximum of 2 instance variables. If not fulfilled: Discussion Rule violation information occurs on the level of the class or on the level of the third instance variable?
- What checks have to be done for the validation?

- Discussion of 'rule dependencies' within one rule (example: wrapper has to determine possible wrapper classes first before beeing able to indicate the use of a primitive/string in a non-wrapper class...)
- –> solution found/no solution found
- Therefore: Be self-critical: Now, were a "solution" is found (or not), describe the problems that occur with the described implementation
- If there are cases where the rule cannot be validated: Why is it hard to validate. Where does the validation fail? How is it possible to "trick" a good implementation (Example: wrap of primitives can be "tricked" with 'return new Wrapper(instancevariable)'...)

### 3.3.1 Validation of Rule 1

Cool, I can draw trees:




### 3.3.2 Validation of Rule 2

asdf

### 3.3.3 Validation of Rule 3

asdf

### 3.3.4 Validation of Rule 4

asdf

### 3.3.5 Validation of Rule 5

asdf

### 3.3.6 Validation of Rule 6

asdf

### 3.3.7 Validation of Rule 7

asdf

### 3.3.8 Validation of Rule 8

asdf

### 3.3.9 Validation of Rule 9

asdf

## 3.4 Result of the Evaluation

=== Give a summary on how hard it was to implement the rules. Where does the rule validator fail. Summarize results of the evaluation. Be positive: Do not forget to emphasize the good and working parts of the evaluation tool and accent the positive outcome of them.

## 3.5 Future Work

=== Be philosophical: What is still to be done in terms of rule validation.

=== If many rules cannot be validated: Future work might be a software where the user can "mark" structures as given. (Example: If it is not possible to determine if a class is a wrapper class, the user could "mark" it as a wrapper class and the validation algorithm might work...)

=== If many rules can be validated: Future work might be the configuration of the rule validator. "pluggable" rules that the user can implement himself? Or "configurable" rule: Use of 2/3/4 instance variables per class.

# 4 Prototypical Implementation of Tool Support

=== This chapter describes the implementation of the prototype.

## 4.1 Requirements

=== Describe requirements for the prototype. Depends on how "good" the prototype is....

## 4.2 Architecture

=== Describe overall architecture.

=== How about a package overview and a short description: What do the classes of the package do and how do they interact in the software?

## 4.3 Exemplary Rule Validation

=== One example implementation of one rule validation. Idea: Implementation of an ASTVisitor class...

## 4.4 Resulting Prototype

=== If the protype is really good: swagger and present it as a good software, that might be used in trianings?

=== If not: Be proud of what is there :). This is only a prototype. What could be improved in a better implementation? Is it even possible to do a better implementation in terms of the rule validation (depends on the outcome of chapter 3).

=== Show screenshot and describe UI. What is possible to do with the client

=== Independent form the quality of the prototype: What are ideas that are still out there for the protoype? How could the product improve?

## 4.5 Outlook and Future Work

=== Possible future work, dependent on what is said in the previous section.

# 5 Conclusion

=== Conclusion of the result of this work. Do not explain in detail, but refer to the Introduction: What was good, what was bad?

TODO: Have fun writing and stay happy :)

# Bibliography


[FoBa03] Foo, John; Bar, Belinda: *Titel : Untertitel*,
Verlagsort: Verlag, Jahr der Auflage. S. 10-20

[Le01] Autor Name: *Titel des Buches*, New York: Penguin Books, 2001.

LITERATUR:

[Bay, 2008] Bay, Jeff: *Object Calisthenics*.
In: ThoughtWorks inc. (eds): *The ThoughtWorks Anthology. Essays on Software Technology and Innovation*.
Raleigh, North California; Dallas, Texas: The Pragmatic Bookshelf, 2008, p. 70–80.

[ThoughtWorks inc., 2008] ThoughtWorks inc. (eds): *The ThoughtWorks Anthology. Essays on Software Technology and Innovation*.
Raleigh, North California; Dallas, Texas: The Pragmatic Bookshelf, 2008.

[Gamma et al., 1994] Gamma, Eric; Helm, Richard; Johnson, Ralph; Vlissides, John: *Design Patterns. Elements of Reusable Object-Oriented Software*.
Amsterdam: Addison-Wesley Longman, 1994.

[Martin, 2008] Martin, Robert Cecil: *Clean Code. A Handbook of Agile Software Craftsmanship*.
n.p., Prentice Hall International, 2008.

[Fowler et al., 1999] Fowler, Martin: *Refactoring. Improving the Design of Existing Code*.
Amsterdam: Addison-Wesley Longman, 1999.

[Evans, 2003] Evans, Eric: *Domain-Driven Design. Tackling Complexity in the Heart of Software*.
Amsterdam: Addison-Wesley Longman, 2003.

asdfklsajkflj INTERNET

[Eclipse Documentation] The Eclipse Foundation: *Eclipse documentation - Eclipse Kepler*.
URL http://help.eclipse.org/kepler/index.jsp.

[Wikipedia] Wikipedia. The Free Encyclopedia.
URL http://www.wikipedia.org.

[ToughtWorks inc.] Thoughtworks inc.
URL thoughtworks.com/about-us (17 November 2013).