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

von

Fabian Schwarz-Fritz

Publication date:	November 16, 2013
Time required for processing:	12 Weeks
Matriculaiton number, course:	123 456, TINF11B2
Vocational training company:	SAP AG, Walldorf
Reviewer:	Softwareschneiderei GmbH

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 16, 2013

Fabian Schwarz-Fritz

Sperrvermerk

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

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"[OC, 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[...]"[TW]

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

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.

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 de-

velopers to focus the rules they automatically apply various helpful and important software principals and software patterns.

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

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 introduciton o fpaper. Qualities, research stuff.

===RESEARCH=== poorly written code, procedural code

Desribe each very shortly:

no reusabilaity

hard to maintain

overview

structure

no bundle o data and behaviour

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

2.1 Advantages of Object Oriented Programming

2.2 The Rules and their Background

This chapter describes the rules of the Object Calisthenics itself.

Repeat form introduciton: The exercise: strict coding standards etc... RESEARCH page 5

Every rule is described one after the other. For every rule there will be ??? infrmation??? . This is a short explanation, a good and a bad example. Furthermore every chapter describes the software patterns and software principals behind every rule. These are for example design patterns, software principals and best practices. The outcome of every rule is then summarized.

For every rule 1-9: - Explain the rule. Use quotes of paper. (research) - Good example with explanation - Bad example with explanation - Describe idea and principal behind the pattern (research). Refer to other sources with large stuff to explain and refer to previous chapter if already explained... - Summarize the rule's purpose

2.2.1 "Use One Level of Indentation per Method"

asdf

2.2.2 "Don't Use the else Keyword"

asdf

2.2.3 "Wrap All Primitives andvlue."

asdf

2.2.4 "Use Only One Dot per Line"

asdf

2.2.5 "Don't Abbreviate"

asdf

2.2.6 "Keep All Entities Small"

asdf

2.2.7 "Don't Use Any Classes with More Than Two Instance Variables"

asdf

2.2.8 "Use First-Class Collections"

asdf

2.2.9 "Don't Use Any Getters/Setters/Properties"

asdf

2.3 Discussing the Rules

2.3.1 Similarities

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

2.3.2 Precedence

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 an outcome of the rules. 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: conslucstion, 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 Tool Support and It's Advantage

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?

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. Describe shortly how it is possible to do an AST validation with eclpise. Say that: "Eclipse" terms and standart terms are use. These are not further described in this paper. Give reference for questions about "parameter", "type", "class", "expression" or "statement". Say that the validaiton in the next section is exactly implemented as described. One (???) example is shown in the Prototype chapter.

3.3 Evaluation of Rule Validation

Say that the priorization of the rules and the "ranking" is given in the end. This chapter is "rule specific", even if the next subsections refer to each other.

Foreach: - Similarities found out in description may be similar in this validation? - Cathegorize the rules in groups form a validation perspecitve - Use examples given in description chapter to describe the typical structure of the rule. - Explane "the positive case": What is the positive structure, satisfying the rule - What checks have to be done for a possible validation - Discussion of 'rule dependencies' within one rule (example: wrapper has to determine possible wrapper classes first...) - –> solution found/no solution found

- Be self-critical: Now, were a solution is found (or not), describe the problems that occur with the described implementation

3.3.1 "Use One Level of Indentation per Method"

asdf

3.3.2 "Don't Use the else Keyword"

asdf

3.3.3 "Wrap All Primitives andvlue."

asdf

3.3.4 "Use Only One Dot per Line"

asdf

3.3.5 "Don't Abbreviate"

asdf

3.3.6 "Keep All Entities Small"

asdf

3.3.7 "Don't Use Any Classes with More Than Two Instance Variables"

asdf

3.3.8 "Use First-Class Collections"

asdf

3.3.9 "Don't Use Any Getters/Setters/Properties"

asdf

3.4 Result of the Evaluation

Give a summary on how hard it was to implement the rules

3.5 Future Work

4 Prototypical Implementation of Tool Support

4.1 Requirements

Describe requirements for the prototype

4.2 Architecture

Describe overall architecture (???packages)

4.3 Exemplary Rule Validation

One example implementation of one rule validation

4.4 Resulting Prototype

Show screenshot and describe UI. What is possible, what is not possible. What are ideas that are still out there? How could the product improve?

4.5 Outlook and Future Work

5 Conclusion

5.1

5.2

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

[OC] Bay, Jeff: *The ThoughtWorks Anthology : Essays on Software Technology and Innovation*,
Raletgh, North California [u.a.]: The Pragmatic Bookshelft, 2008

[TW] Thoughtworks inc. URL: thoughtworks.com/about-us