Validating the Object Calisthenics

Evaluation and Prototypical Implementation of Tool Support

Jeff Bay's „Object Calisthenics" are exercises to improve the quality of object oriented code. Good object oriented code is hard to learn when coming from procedural code. Many developers think object oriented – but do they really write good object oriented software?

The rules of the Object Calisthenics are presented in *The ThoughtWorks Anthology* by Jeff Bay. The rules train developers to enhance their object oriented coding style. The calisthenics are composed of nine rules that the developer has to stick with. Behind every rule there is a purpose why the rule is important and why it leads to better object oriented code.

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.

But when completing the training challenge the developer has to observe his own code and check if his coding satisfies the nine rules of the Object Calisthenics. Tool support could shorten the time of the training and furthermore guarantee that the developer sticks to the given rules.

In a student research paper I am currently evaluating the development of tool support for the Object Calisthenics. In the course of the paper I already implemented a prototype. The paper is still in process. The tool created during the research of the paper is realized in form of an Eclipse plugin. It is successfully validating the majority of Jeff Bay's rules and indicates corresponding violations.

This contribution consists of two parts.

The first part compromises the short explanation of patterns and principals behind the rules.

The challenges validating the compliance are the second and main part of this contribution. Is it possible to validate the rules? Can the rules be categorized? Are there similarities in validating the rules? Can the rules be grouped in different categories? What challenges occurred when validating the source code structure? For which rules was it not possible to find an implementation, determining the validity of a rule and what detained it? In this part the implementation of at least one rule validation is exemplified.