Assuring Software Quality by Code Smell Detection

(Invited talk, Most Influential Paper Award)

Eva van Emden
Vancouver Editor
Vancouver, BC, Canada
Email: eva@alumni.uvic.ca

Leon Moonen*
Simula Research Laboratory
Lysaker, Norway
Email: leon.moonen@computer.org

***Abstract*—In this retrospective we will review the paper "Java Quality Assurance by Detecting Code Smells" that was published ten years ago at WCRE. The work presents an approach for the automatic detection and visualization of code smells and discusses how this approach could be used in the design of a software inspection tool. The feasibility of the proposed approach was illustrated with the development of jCOSMO, a prototype code smell browser that detects and visualizes code smells in JAVA source code. It was the first tool to automatically detect code smells in source code, and we demonstrated the application of this tool in an industrial quality assessment case study.**

In addition to reviewing the WCRE 2002 work, we will discuss subsequent developments in this area by looking at a selection of papers that were published in its wake. In particular, we will have a look at recent related work in which we empirically investigated the relation between code smells and software maintainability in a longitudinal study where professional developers were observed while maintaining four different software systems that exhibited known code smells. We conclude with a discussion of the lessons learned and opportunities for further research.

***Index Terms*—software inspection, quality assurance, Java, refactoring, code smells.**

SUMMARY

Software inspection is a known technique for improving software quality that involves carefully examining the code, design, and documentation of a system to check for aspects that are known to be potentially problematic based on past experience. Given that the cost of repairing defects is much lower at the beginning of the development cycle, one of the main advantages of software inspection is that it does not require execution, thereby enabling early identification of potential problems, when it's still cheap to fix them.

Traditional software inspection is a systematic and disciplined process that involves labor-intensive manual phases. The strict requirements often backfire, resulting in ill-performed or abandoned inspection processes. Our paper was part of a body of work that aimed to alleviate these challenges by automating the software inspection process using tools that can detect potential issues via source code analysis. Automatic inspection of code quality and conformance to coding standards allows early and repeated detection of signs of project deterioration, which in turn enables early corrections, lowers the development costs and increases the chances for success.

In 2002, most tools that supported automatic code inspection took a technical perspective in which code inspections boiled down to bug-chasing, with tools looking for problems with pointer arithmetic, memory (de)allocation, `null` references, array bounds errors, etc. Our work took a completely different perspective: inspired by the metaphor of "*code smells*" introduced by Beck and Fowler, we reviewed code for problems that are associated with bad program *design* and bad programming *practices*. Code smells were originally introduced to describe characteristics of code that indicate that it can be refactored. "*Refactoring is the process of changing a software system in such a way that it does not alter the external behavior of the code yet improves its internal structure*". Refactoring tidies up source code and reduces its complexity, so that the system is easier to understand and maintain.

Our WCRE 2002 paper did not use code smells to guide refactoring, but instead was the first to investigate *feasibility of automatic code smell detection* via source code analysis, and examined how such automatic code smell detection could contribute to automatic software inspection by *assessing software quality attributes* such as understandability and maintainability. The retrospective will discuss these and subsequent developments in the area of code smell detection by looking at a selection of papers that were published in its wake. In particular, we will have a look at recent related work in which we empirically investigated the relation between code smells and software maintainability in a longitudinal study where professional developers were observed while maintaining four different software systems that exhibited known code smells. We conclude with a discussion of the lessons learned and an overview of opportunities for further research in this field.

Acknowledgment: The research described in the WCRE 2002 paper was conducted at Centrum voor Wiskunde en Informatica (CWI), the Netherlands. Eva van Emden was a visiting research assistant from the Rigi Group, Computer Science Department, University of Victoria, Victoria, BC, Canada.

About the authors: Eva van Emden received her MSc in Computer Science from University of Victoria, Canada. She is currently a freelance editor with a special interest in editing academic and popular science writing. Eva can be found online at vancouvereditor.com.

Leon Moonen is a senior research scientist at Simula Research Laboratory in Norway. His research is aimed at creating better tools for the exploration, assessment and evolution of large industrial software systems. This combines several subfields of software engineering, such as program comprehension, reverse engineering, program analysis, software visualization and empirical software engineering. Leon's work can be found online at leonmoonen.com.

*Corresponding author.