inCode: Continuous Quality Assessment and Improvement

Radu Marinescu, George Ganea and Ioana Verebi

LOOSE Research Group
Politehnica University of Timișoara, Romania

Abstract—While significant progress has been made over the last ten years in the research field of quality assessment, developers still can’t take full advantage of the benefits of these new tools and technique. We believe that there at least two main causes for this lack of adoption: (i) the lack of integration in mainstream IDEs and (ii) the lack of support for a continuous (daily) usage of QA tools. In this context we created INCODE as an Eclipse plugin that would transform quality assessment and code inspections from a standalone activity, into a continuous, agile process, fully integrated in the development life-cycle.

But INCODE not only assesses continuously the quality of Java systems; it also assists developers in taking restructuring decisions, and even supports them in triggering refactorings.

I. INTRODUCTION

The typical usage scenario of a QA module/methodology is currently this: a developer, feeling that something is wrong with the design/code, is using the QA module provided by (or available for) her IDE to compute a suite of metrics; noticing some abnormal metric values, she must infer what the real design problem is from the informal description of the metric’s interpretation model. This is not easy at all, especially when the analysis occurs long after that code/design fragment has been created, and/or the code was written by someone else. But even after finding out what the problem is, correcting the design flaws moves the developer to another world, where she must compose the proper restructuring solution using the basic refactorings available in her IDE.

This is again a challenging and painstaking operation. We believe that this process is so tedious because of two reasons: (i) metrics used to detect design flaws are only “detection atoms”, and, therefore incapable of pointing out to relevant correction (restructuring) solutions; (ii) refactorings, as they are used know, are also only the “correction atoms”, and therefore they don’t represent the correction solution for all but non-trivial design problems.

With INCODE we propose a novel type of QA tools, with the following (agile) “manifesto”: (i) Continuous detection and correction of design flaws, instead of standalone, post-factum (or even “post-mortem”) code inspections; (ii) Exposure of real design flaws, instead of displaying abnormal metrics values; (iii) Contextual, problem-driven, tool-assisted restructuring strategies, instead of a repetitive (ad-hoc) composition of restructuring solutions from atomic refactorings. Next we are going to briefly introduce (some of) the main features of INCODE i.e., those that reveal its novelty.

II. CONTINUOUS DETECTION OF DESIGN PROBLEMS

INCODE is an Eclipse plugin in which design problems are detected continuously. When the Eclipse workbench starts, INCODE also starts to analyze (in background) the source file currently active in the editor. When a design problem is detected, a red marker is placed on the ruler, next to the affected class or method (and also in the overview ruler on the right side of the editor). The inCode markers are similar to the markers used for compiler errors or warnings (see Figure 1). The presence of these markers is very dynamic: as new code is written, or code is modified new markers may appear, or existing markers may disappear.

As mentioned before, we believe that metrics used in isolation cannot help in detecting real design problems. Therefore, in INCODE we use detection strategies [1] to quantify design problems [2]. Currently INCODE detects four design problems related to an improper distribution of intelligence among classes. Specifically these are: God Class, Data Class, Feature Envy and Code Duplication [3], [4]. Thus, while the detection is based on object-oriented metrics, developers don’t have to interact directly with metrics. Instead, they can reason about the quality of their design at the conceptual level that is more convenient for them.

III. CAUSES AND CURES FOR EACH DESIGN PROBLEM

In order to see the details about what caused a class/method to be detected as having a particular design problem, the user clicks on the red marker and selects the
design problem of interest. This action opens the inCode Tips view in the lower part of the workbench, providing the developer with two sorts of information: (i) contextualized hints about why the inspected class or method has that particular design problem; and (ii) concrete advices (named Refactoring Tips, see Figure 2) on how the problem could be corrected, by taking into account the entire context of dependencies in which the class/method is involved. In particular, when INCODE detects that in the given context a predefined Eclipse refactoring can be applied, the refactoring (e.g., Move Method) can be triggered directly from the inCode Tips view. Currently, we are working on extending INCODE with the possibility to trigger more complex restructurings that involve the correlation of multiple atomic refactorings.

IV. SYSTEM/PACKAGE OVERVIEWS

In addition to continuous detection of design problems, INCODE offers an overview (both at the system and package level), which can be used, for example, as a starting point for an initial code inspection. The overview has two components: (i) the Overview Pyramid [2] that captures the key characteristics of the system/package in terms of complexity, coupling and shape of class hierarchies; (ii) a categorized list of detected design problems. From here, the problematic classes and methods can be inspected closer in order to understand for each cases the particular causes and the suggested correction steps. Due to the close integration with Eclipse, at any moment, the engineer has direct access from INCODE to the source-code of the problematic code fragment.

V. MORE FEATURES

INCODE assesses continuously the design quality of Java systems, and assists developers in solving the occurring problems by providing contextual refactoring hints. Additionally, INCODE supports also quality assessment at the architectural level by detecting and providing contextual information about three well-known architectural problems: cyclic dependencies, unstable dependencies, and the lack of dependency on abstractions [5]. INCODE supports code understanding by serving developers, in a context-sensitive manner, with interactive polymetric visualizations [2].

VI. DOWNLOAD AND INSTALL

INCODE is very easy to install using the standard Eclipse mechanism for plugin installation. More information can be found at http://www.intooitus.com/inCode.html. INCODE can be used for both commercial and non-commercial projects. So far, we successfully used INCODE to projects up to 1.4 MLOC. We would appreciate very much if you could let us know about using INCODE. In return we would do our best to provide you with support and updates.

ACKNOWLEDGMENT

We gratefully acknowledge the financial support of the Romanian National Science Foundation (CNCSIS) for the project “Methods and Tools for Continuous Quality Assurance in Complex Software Systems” (PNII-IDEI 357/1.10.2007).

REFERENCES


