2017 IEEE 25th International Conference on Program Comprehension (ICPC)

FindSmells: Flexible Composition of Bad Smell Detection Strategies

Bruno L. Sousa, Priscila P. Souza, Eduardo Fernandes
Computer Science Department
UFMG - Belo Horizonte, Brazil
Email: {brunosousa, priscilasouza, eduardofernandes}@dcc.ufmg.br

Kecia A. M. Ferreira
Computer Department
CEFET-MG - Belo Horizonte, Brazil
Email: kecia@decom.cefetmg.br

Mariza A. S. Bigonha
Computer Science Department
UFMG - Belo Horizonte, Brazil
Email: mariza@dcc.ufmg.br

***Abstract*—Bad smells are symptoms of problems in the source code of software systems. They may harm the maintenance and evolution of systems on different levels. Thus, detecting smells is essential in order to support the software quality improvement. Since even small systems may contain several bad smell instances, and considering that developers have to prioritize their elimination, its automated detection is a necessary support for developers. Regarding that, detection strategies have been proposed to formalize rules to detect specific bad smells, such as Large Class and Feature Envy. Several tools like JDeodorant and JSpIRIT implement these strategies but, in general, they do not provide full customization of the formal rules that define a detection strategy. In this paper, we propose FindSmells, a tool for detecting bad smells in software systems through software metrics and their thresholds. With FindSmells, the user can compose and manage different strategies, which run without source code analysis. We also provide a running example of the tool.**

Video: https://youtu.be/LtomN93y6gg

I. Introduction

Bad smells are symptoms of problems in the source code or design of software systems [1]. These bad smells may affect the code of a system on different levels, such as packages, classes, and methods [2]. For instance, *Large Class* is a class with excessive knowledge and responsibilities [1] that hinders the code readability, due to its extent and its structural complexity. Changes in a *Large Class* may affect several different parts of the system. Thus, it is necessary to detect bad smells in order to improve the quality and decrease the maintenance efforts in software development [3], [4].

Even small-sized software systems may have several bad smell instances [5]. Thus, finding smells requires too much efforts from the development team. In addition, the perception of developers regarding each smell varies from one developer to another. This happens because manual inspection is subjective, i.e., it depends on the developer's expertise in bad smells and their definitions [6]. In turn, formal, well-defined strategies can support the automated detection of bad smells without human-related biases. In this context, *detection strategies* are formal rules aimed at characterizing bad smells by combining software metrics, thresholds, and logical operators [7].

Previous work [8], [9] proposes detection strategies for different bad smells such as *Data Class*, *Feature Envy*, and *Long Method*. To implement these strategies, some detection tools have been proposed [2], [10]. However, several metric-based tools do not allow detection strategies to be configured or customized. Therefore, a tool based on computed metrics, with flexible configuration of detection strategies, may guide the developers in the detection phase and, then, in the priorization of bad smells for elimination.

This paper presents a tool, FindSmells, which supports the detection of bad smells based on software metrics and their thresholds. The tool receives as input XML files containing the software metrics computed for a software system. With FindSmells, the user may propose its own detection strategies considering the software metrics available in the XML files, the comparison logical operators, and the thresholds per metric. In order to assess whether the tool is able to support both composition and run the detection strategies we conducted a study with 12 Java software systems from Qualitas.class Corpus [11]. Our results suggest that FindSmells may be effectively used to support bad smell detection in an easy way.

II. Bad Smell Detection Strategies

A *detection strategy* is a formal rule that charatecterizes a specific bad smell. In addition, previous work [2], [7] defines detection strategies for different bad smells, such as *Large Class* and *Feature Envy*. Each detection strategy is composed by a sequence of clauses connected by the logical operators AND and OR, with the operator AND preceeding OR in the definition of strategies, including thresholds. A clause is composed by a software metric, e.g., Method Number of Code Lines (MLOC), a comparative operator, e.g., $>$, and a threshold, e.g., 1000. For instance, the clause $MLOC > 1000$ may support the identification of a *Long Method* [1], a method that is excessively large and complex.

A key factor to compose effective detection strategies is the selection of representative thresholds [12]. Concerning this, previous work [12], [13], [14] proposes different techniques to derive thresholds for software metrics and, also, catalogs of thresholds for different sets of metrics [4].

Some supporting tools [6], [10], [15] provide the automatic running of detection strategies. However, they are not flexible with respect to the customization of clauses, or regarding threshold changing. Eventually, some changes are necessary

978-1-5386-0535-6/17 $31.00 © 2017 IEEE
DOI 10.1109/ICPC.2017.8

to apply a detection strategy in a specific development context because systems from different domains tend to vary in size, complexity, and programming language. Therefore, the customization of detection strategies may benefit developers in the software quality assessment.

III. The FindSmells Tool

This section presents the proposed tool, FindSmells. FindSmells' approach uses a static analysis of bad smells based on computed metrics and its thresholds. This decision was taken to reduce the response time when analyzing large systems with hundreds of source elements, such as packages, classes, and methods. Although software metrics have to be computed to support the analysis performed by FindSmells, well-known and largely used tools such as Metrics[1] and CodePro Analytix[2] support this task. Our goal is to support developers with a flexible, light-weighted mean to assess the software quality by the identification of bad smells that may harm the maintenance and evolution of software systems.

The section remaining enlist in Section III-A the main features of FindSmells. Section III-B describes the tool's architecture. Section III-C provides technical data in details regarding its implementation.

A. Main Features

The main features of FindSmells are described as follows.

Import of Computed Metrics. Identify bad smells in a given system, FindSmells requires that the user imports an XML file with computed software metrics for the system. This input file has to be written in accordance with a well-defined format described on the tool's website [16].

Composition of Detection Strategies. The user may compose new detection strategies through (i) a panel with the 21 software metrics computed by the Metrics or other tool, (ii) a comparative logical operators ($<$, $\leq$, $>$, $\geq$, and $=$), and (iii) thresholds defined by the user. To guide users in the definition of thresholds, FindSmells suggests a catalog proposed by Filó et al. [4]. This catalog provides thresholds for 18 software metrics such as McCabe's Cyclomatic Complexity, Number of Methods (NOM), and Weighted Methods per Class (WMC). This catalog is available in the Help menu in FindSmells, and it was previously evaluated by means of empirical studies [4], [17]. Each composed strategy may be saved in the FindSmells.

Management the Detection Strategies. The user may consult previously saved detection strategies in order to run them again as it is, or he may replace only the thresholds. In case the user needs to configure the strategy to change metrics and logical operations, a new strategy must be composed. The user also can delete a detection strategy from the database.

Running the Detection Strategies. To run the detection strategy defined by the user, FindSmells filters the source code elements, methods and classes, that are in accordance with the rules defined for each metric composing the detection strategy.

[1]http://metrics.sourceforge.net/

[2]https://marketplace.eclipse.org/content/codepro-analytix

Result Generation, Visualization, and Export. After running the detection strategy, FindSmells generates a report with all the occurrences of the respective bad smell. The obtained results are presented to the user in a data grid view for navigation. The tool also exports the obtained results via CSV file to provide easy manipulation and further analysis. This output file is automatically saved in the tool's workspace.

Log Visualization. The user may consult logs of the tool, in order to identify problems that eventually occur in the imported XML file. This feature may be helpful to track some problems like invalid characters in the input file.

Help. To support the users that are unfamiliar with thresholds, FindSmells provides a catalog proposed by Filó et al. [4] with 18 metrics thresholds for consultation. This catalog may assist the users in the composition of detection strategies. In this menu, the user may consult the link for a video with the main features of FindSmells.

B. Tool's Architecture

Figure 1 illustrates the architecture of FindSmells. The tool is composed by five modules, whose functionalities are described as follows.

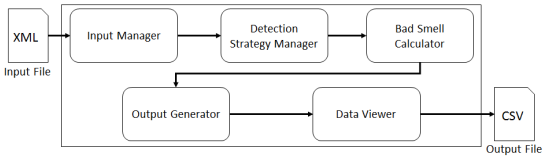

Fig. 1. Architecture of FindSmells

Input Manager. This module is responsible for the input management. It receives one or more XML files with precomputed software metric values for a given system and reads these files according to the proposed input format. The pattern XML format follows the format exported by Metrics. Since FindSmells performs a static analysis from metrics, it supports bad smell detection in projects developed in any programming language. This module also assesses the occurrence of inconsistences in the input file, such as invalid characters, and generates an error report that can be exported for further analysis. Then, the metric values are persisted in the database. The default input format is available for consultation in the tool's website [16].

Detection Strategy Manager. This module provides the composition, edition, and exclusion of detection strategies by the user. It implements a validator to assess whether the logical expression composing the detection strategy is syntactically correct. For instance, it validates the correct use of parentheses and the use of logical connectors. In addition, the composed strategies are persisted in the database.

Bad Smell Calculator. This module implements the execution of a given detection strategy on the software metrics persisted in the tool's database. It filters the source code elements in

accordance with the detection strategy definition, in order to identify bad smells occurrences.

Output Generator. From the list of bad smells occurrences computed by FindSmells, this module generates a CSV output file depending on the granularity of the detection strategy, which is, class, method, or package-level. The output file contains the name of each source code that is affected by a bad smell, the element's source – in the case of method-level strategies, the source is a class – and the element's source package identified by the detection strategy. These data may be helpful to support the smell traceability.

Data Viewer. This module provides the output report via data grid view, allowing the user to navigate through the list of bad smell occurrences identified. In addition, it presents the rate of code elements affected by a specific bad smell.

C. Implementation

FindSmells was implemented in Java programming language with support of JDK 1.7 and the Java Swing API for creating the user interface. We chose Java because it is one of the most popular languages, in academy and industry. To provide data persistance, we adopted the SQLite[3] database manager, Release 3.8.11.2, and the JDBC API, because it is light-weighted and easy to integrate with the source code. To parse XML input file, we used the JDOM API that provides XML interpretation for Java applications. To build the user interface, we used the NetBeans IDE[4], release 8.0.2, which provides a drag and drop feature for this purpose. FindSmells is available in Release 1.0 via research website [16]. The source code contains 6,049 lines of code (LOC).

IV. Running Example

This section describes the FindSmells' usage. Figure 2 presents the main screen of FindSmells. This screen provides three functionalities: (i) in the module *Import System*, the user imports the XML input file with the software metrics for a single system. For each imported system, the user can define a name for future identification; (ii) in *Run Detection Strategy*, the user selects a previously imported system for analysis and a previously composed detection strategy to run; (iii) in the top of the main screen appears a menu composed of four options: *File* to view error reports; *Manage Strategies* to compose, edit, and delete detection strategies; *Manage Systems* to delete imported systems; and *Help*, which contains a reference of threshold catalog to support the strategy composition and an *About* option with the tool's release, developer list, and links for the source code and a video tutorial.

Figure 3 presents the screen for composing detection strategies. It is possible to define a name for the strategy and, also, its granulatiry. Depending on the selected granularity, a specific set of metrics is unlocked for selection. To compose

[3]https://sqlite.org/
[4]https://netbeans.org/

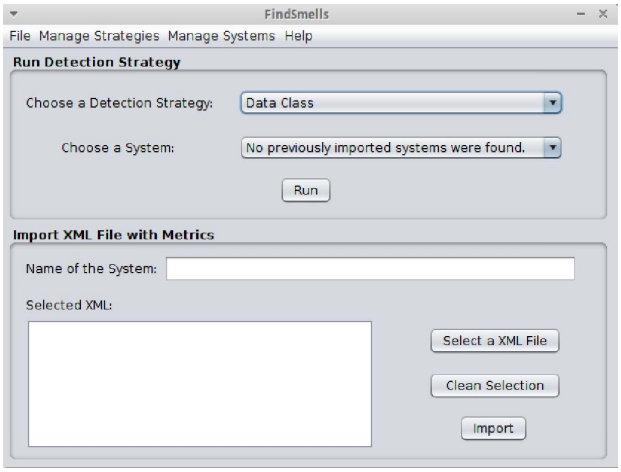

Fig. 2. Main Screen to Import XML Files and Run Detection Strategies

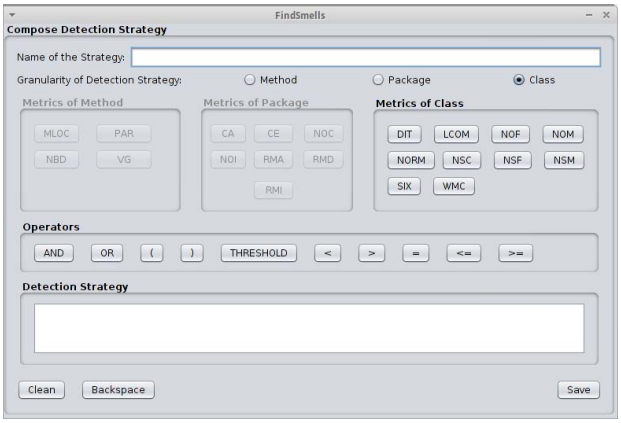

Fig. 3. Screen to Compose a Detection Strategy

a clause, the user may select logical operators, separators, e.g., parentheses. During the detection strategy composition, its components are presented in the box *Detection Strategy*. Then, the tool presents a list of metrics used to compose the proposed strategy. The user must manually provide the thresholds per metric in the next screen after saving the detection strategy.

After running the detection strategy selected in the screen of Figure 2, FindSmells presents the list of bad smell occurrences identified by the strategy. Finally, Figure 4 illustrates the data grid view with the bad smell list, allowing the user navigate through the detection results. Each line of the grid contains the same columns reported by the CSV output file (see Section III-B). In addition, there is a button in the inferior corner of the screen to export the detection results.

V. Tool's Evaluation

FindSmells was evaluated in a empirical study conducted by Souza [17]. For this purpose, the user registered and run detection strategies for 5 bad smells, namely *Large Class*, *Long Method*, *Data Class*, *Feature Envy*, and *Refused Bequest*.

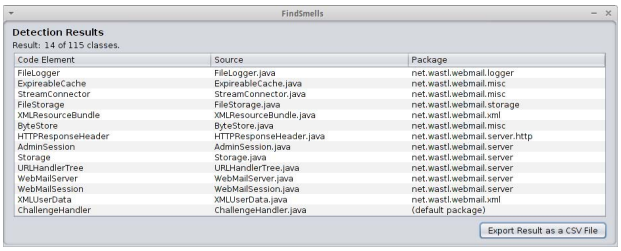

Fig. 4. Data Grid View and Button to Export Results as a CSV File

The composed strategies used are based on previous work [1], [2]. Each strategy was executed with 12 Java systems from Qualitas.class Corpus [11]. The results suggest that FindSmells is able to run correctly the strategies proposed by the user [17]. The evaluation results are available in the tool's website [16].

VI. Related Work

Several tools, for instance JSpIRIT [6] and JDeodorant [10], for bad smell detection have been proposed in the literature. JDeodorant is an Eclipse IDE plug-in tool for Java software systems that aims to detect a subset of bad smells listed by Fowler [1], such as *Large Class*, *Feature Envy*, and *Switch Statement*. The tool provides data exported for further analysis. PMD[5] is a tool largely used in academic research. It provides the detection of several bad smells, including *Large Class*, *Long Method*, and *Duplicated Code*. JSpIRIT is a tool for Java systems, also an Eclipse plug-in with the data export feature. This tool aims to detect a larger set of bad smells, such as *Long Method*, *Data Class*, and *Shotgun Surgery*.

In addition to the large amount of proposed tools, there are different techniques used to support these tools. Most of them are metric-based [18] but their detection strategies are predefined without high flexibility for configuration. Machine learning, lexical analysis, and graphs are other approaches used to support the detection of bad smells. However, many of them may not scale for medium to large-sized systems, with several methods, classes, and packages [3], [5]. We developed FindSmells, which provides flexibility to the user regarding the composition of detection strategies from a set of computed metrics and their thresholds, addressing medium to large systems with several methods, classes and packages.

VII. Conclusion

This paper presents FindSmells, a tool for detecting bad smells. This tool receives as input the XML files containing computed software metrics. Through a simple user interface, the tool allows the user to compose detection strategies and run them without the source code of the system under analysis. FindSmells also allows the user to manage detection strategies through a database, and provides the exportation of detection results. FindSmells is a tool whose purpose is to support the development team in assessing the quality of software systems.

[5]https://pmd.github.io/

Acknowledgments

This work was sponsored by CAPES.

References

[1] M. Fowler, *Refactoring: Improving the Design of Existing Code*. Pearson Education, 2009.
[2] M. Lanza and R. Marinescu, *Object-Oriented Metrics in Practice*. Springer, 2006.
[3] S. Bellon, R. Koschke, G. Antoniol, J. Krinke, and E. Merlo, "Comparison and Evaluation of Clone Detection Tools," *IEEE Transactions on Software Engineering (TSE)*, vol. 33, no. 9, pp. 577–591, 2007.
[4] T. Filó, M. Bigonha, and K. Ferreira, "A Catalogue of Thresholds for Object-Oriented Software Metrics," in *Proc. of the 1st SOFTENG*, 2015, pp. 48–55.
[5] I. Macia, J. Garcia, D. Popescu, A. Garcia, N. Medvidovic, and A. von Staa, "Are Automatically-Detected Code Anomalies Relevant to Architectural Modularity?: An Exploratory Analysis of Evolving Systems," in *Proc. of the 11th AOSD*, 2012, pp. 167–178.
[6] S. Vidal, C. Marcos, and J. Díaz-Pace, "An Approach to Prioritize Code Smells for Refactoring," *Automated Software Engineering (ASE)*, pp. 1–32, 2014.
[7] R. Marinescu, "Measurement and Quality in Object-Oriented Design," in *Proc. of the 21st ICSM*, 2005, pp. 701–704.
[8] ——, "Detection strategies: Metrics-based rules for detecting design flaws," in *Software Maintenance, 2004. Proceedings. 20th IEEE International Conference on*. IEEE, 2004, pp. 350–359.
[9] M. J. Munro, "Product metrics for automatic identification of" bad smell" design problems in java source-code," in *Software Metrics, 2005. 11th IEEE International Symposium*. IEEE, 2005, pp. 15–15.
[10] N. Tsantalis, T. Chaikalis, and A. Chatzigeorgiou, "JDeodorant: Identification and Removal of Type-Checking Bad Smells," in *Proc. of the 12th CSMR*, 2008, pp. 329–331.
[11] R. Terra, L. F. Miranda, M. T. Valente, and R. Bigonha, "Qualitas.class Corpus: A Compiled Version of the Qualitas Corpus," *ACM SIGSOFT Software Engineering Notes*, vol. 38, no. 5, pp. 1–4, 2013.
[12] K. Ferreira, M. Bigonha, R. Bigonha, L. Mendes, and H. Almeida, "Identifying Thresholds for Object-Oriented Software Metrics," *Journal of Systems and Software (JSS)*, vol. 85, no. 2, pp. 244–257, 2012.
[13] T. Alves, C. Ypma, and J. Visser, "Deriving Metric Thresholds from Benchmark Data," in *Proc. of 26th the ICSM*, 2010, pp. 1–10.
[14] P. Oliveira, M. T. Valente, and F. P. Lima, "Extracting Relative Thresholds for Source Code Metrics," in *Proc. of the CSMR-WCRE*, 2014, pp. 254–263.
[15] E. Murphy-Hill and A. Black, "An Interactive Ambient Visualization for Code Smells," in *Proc. of the 5th SOFTVIS*, 2010, pp. 5–14.
[16] B. Sousa, P. Souza, E. Fernandes, M. Bigonha, and K. Ferreira, "FindSmells," 2016, Available at: http://www2.dcc.ufmg.br/laboratorios/llp/Products/indexProducts.html. (accessed November 17, 2016).
[17] P. Souza, "Using Software Metric Thresholds to Assess the Quality of Object-Oriented Software Systems," 2016, MSc. dissertation, Federal University of Minas Gerais, Belo Horizonte, Brazil. (in portuguese).
[18] E. Fernandes, J. Oliveira, G. Vale, T. Paiva, and E. Figueiredo, "A Review-Based Comparative Study of Bad Smell Detection Tools," in *Proc. of the 20th EASE*, 2016, pp. 1–12.