Santos *et al. Journal of Software Engineering Research and Development* (2017) 5:8
DOI 10.1186/s40411-017-0042-0

Journal of Software Engineering Research and Development

RESEARCH Open Access

Investigating factors that affect the human perception on god class detection: an analysis based on a family of four controlled experiments

José Amancio M. Santos[1*], João B. Rocha-Junior[2,3] and Manoel Gomes de Mendonça[4,3]

*Correspondence:
zeamancio@uefs.br
[1]Technology Department, State University of Feira de Santana, Avenida Transnordestina, s/n - Novo Horizonte, CEP 44036-900, Feira de Santan, Bahia, Brazil
Full list of author information is available at the end of the article

Abstract

Context: Evaluation of design problems in object oriented systems, which we call code smells, is mostly a human-based task. Several studies have investigated the impact of code smells in practice. Studies focusing on human identification of code smells have shown low agreement among developers. Unfortunately, those studies do not attempt to investigate the reasons behind this phenomenon.

Objective: This paper aims to investigate factors affecting human perception of code smells. Specifically, it focuses on factors affecting god class detection, one of the most known code smells.

Method: The investigation encompassed a family of four controlled experiments, covering potential factors affecting human detection of code smells. The method is incremental. In other words, each experiment produces insights to the next one. This allows the investigators to control specific factors affecting the agreement on god class detection. The factors addressed in this study are: i) developer experience, ii) developer knowledge, iii) developer training, iv) tool support for design comprehension, and v) software size.

Result: Our findings show that tool support for design comprehension is the only factor that does not affect the human perception of god class. The other factors impact this perception in some way.

Conclusion: The area still needs more investigation and discussion on what we call the *code smell conceptualization problem*, to ensure similar criteria and thresholds on human-based code smell detection.

Keywords: Controlled experiment, Code smell, God class

1 Background

The software engineering community has been discussing strategies for the systematic evaluation of potential design problems. This issue was raised as early as 1996 by Riel (1996). He presented insights into design improvement and introduced the term "design flaw" in his book. Fowler (1999), who coined the term "code smell", focused on refactoring and presented a catalog of smells, characterizing and proposing specific actions to remove

them. Following suit, Lanza and Marinescu (2005) focused on metrics and heuristics to detect what they called "disharmonies".

Although, the terms "design flaw", "code smell" and "disharmony" have been used to define potential design problems, this work adopts the term *code smell*, or simply *smell*, to refer to such problems. The works discussing design problems are based on common (and expected) principles such as inheritance, information hiding or polymorphism (Meyer 1988). The authors consider that problems in design occur when these principles are broken. Then, they propose different strategies for code smell detection, which is fundamental as it affects common activities of software development, such as diagnostics in code inspection or refactoring and maintenance decisions.

In practice, all code smell detection strategies are human-based. For example, Fowler (1999) does not provide objective criteria to identify code smells. He says that one needs to develop its own sense of observation about attributes that could characterize pieces of code as a smell. One has, for example, to develop its own sense of how many lines of code define a long method. Lanza and Marinescu (2005) used an objective definition for code smell detection based on metrics and thresholds. However, metrics and thresholds are domain dependent, again introducing subjectivity in the code smell detection process. Rapu et al. (2004) reinforced this observation declaring that metrics thresholds are mainly chosen based on the experience of the analysts. One may then speculate that code smell detection is a subjective task by nature, as its adoption is dependent on human evaluation.

Understanding which and how subjective aspects affect code smell detection demands empirical evaluation. Recently, some empirical studies have been carried out to better understand this scenario (Yamashita and Moonen 2013; Palomba et al. 2014; Linares-Vásquez et al. 2014; Ahmed et al. 2015; Fu and Shen 2015). Despite this, the area lacks studies addressing the human role on smell detection. In a systematic mapping study, Zhang et al. (2011) noted that most studies related to code smells focus on tools and methods for automatic detection (tool assessment category). In a complementary remark, Mäntylä and Lassenius (2006b) declared that the role of humans has been little studied. In fact, many authors reinforce the idea that more empirical studies are necessary to enhance the understanding of the code smell effect (Schumacher et al. 2010; Zhang et al. 2011; Sjøberg et al. 2013).

The human perception has been pointed as a common problem in some studies focusing on the role of humans in code smell detection. Palomba et al. (2014) observed that, for some smells, the perception of the developers varies. Mäntylä and Lassenius (2006a) highlighted the conflicting perceptions of different developers using code smells to evaluate software quality. Schumacher et al. (2010) found low agreement among developers on code smell detection.

Due to these evidences, we believe that, before proposing new strategies, it is necessary to understand the reasons behind the inconsistencies found in the current code smell detection strategies. Thus, our main objective is to improve the understanding of factors affecting the code smells detection. In order to achieve this objective, we performed a family of controlled experiments. We adopt the idea that a family of experiments is formed by a set of experiments presented in a similar context (Basili et al. 1999). The experiments were designed to investigate a number of factors, which potentially affect the human perception of code smells. We ran four controlled experiments, up to now.

For simplicity, we address god class, one of the most known code smells (Fernandes et al. 2016; Zhang et al. 2011).

In this paper, we focus on the investigation of five factors and how they impact the human perception on detecting god class. The factors are *design comprehension tool support*, *developer experience*, *developer knowledge*, *developer training*, and *software size*. Here, we highlight two of our findings. First, we find out that all factors, except design comprehension tool support, affects the human perception of god classes. This supports support our hypothesis that code smell detection is more related to human traits than to an overall understanding of the code design. The idea behind this hypothesis is that, considering the smell detection activity, subjectivity should be more relevant than the understanding of the code design. Second, a training based on examples and group discussion might to mitigate divergences on human perception of god classes and, possibly, of other code smells. From this finding, we conjecture that there is no "silver bullet" in terms of heuristics for smells detection, once benefits adopting the concept are context-dependent.

We have already published partial results of our family of controlled experiments. In these previous works, we covered other aspects on god class detection adopting different strategies of analyses, which were not based on the investigation of factors impacting the human perception on god class detection. In (Santos et al. 2013), we explored the differences among developers detecting god class, in terms of effort and decision drivers. In (Santos et al. 2014), we investigated how a design comprehension tool impacts the effort and the strategies adopted by developers detecting god class. We also deepen the analyses of the strategies (Santos and Mendonça 2014) and the decision drivers (Santos and Mendonça 2015) adopted by developers detecting god class. In summary, we previously explored effort, strategies and decision drivers on god class detection. In this current work, the five factors we cover are different. They represent more empirical evidence on investigation of factors that affect the human perception of god classes.

The structure of the rest of this paper is as follows. In Section 2, we present concepts and summarize prior empirical studies addressing code smell. In Section 3, we introduce the family of controlled experiments. We also present the planning and execution of the experiments. In Section 4, we discuss the factors we addressed in this paper, how we controlled them through the family of experiments and our strategy of analysis. In Sections 5 and 6, we present the results and a discussion about them. In Section 7, we discuss the threats to the validity of the studies. Lastly, in Section 8, we present our conclusions and future work.

2 Context and related work

In this section, we present god classes as the central concept of our experiments. We also present the most relevant studies focusing on code smell evaluation and detection agreement.

2.1 The god class code smell

The term god class was coined by Riel (1996) to refer classes that tend to centralize the intelligence of the system. Since then, god class has been addressed in different empirical studies (Li and Shatnawi 2007; Olbrich et al. 2010; Abbes et al. 2011; Padilha et al. 2013). Riel presented god class as a problem of system intelligence poorly distributed.

The problem manifests itself in the behavioral form when developers "attempt to capture the central control mechanism so prevalent in the action-oriented paradigm within their object-oriented design". The heuristics proposed by Reil to avoid the god class smell are:

- "Top-level classes in a design should share work uniformly. [...]"
- "Beware of classes with much non-communicative behavior. [...]"
- "Beware of classes that access directly data from other classes."

Fowler (1999) did not use the term god class to describe code smells. However, he presented a code smell with similar characteristics. He defined the large class code smell as a class that tries to do too much. For him, a "class with too much code is prime breeding ground for duplicated code, chaos, and death". He proposes the use of the refactoring techniques *extract class*, *extract subclass* and *extract interface* to solve the problem.

Lanza and Marinescu (2005) proposed a heuristic for god class detection. They consider that if a class uses more than few attributes of other classes and has high functional complexity and has low cohesion, then it is a god class. They define the thresholds for few attributes, high functional cohesion and low cohesion according to the software characteristics. Lanza and Marinescu (2005) based it on the definition of Riel (1996) and explicitly declared that the concept is comparable to Fowler's large class smell. The authors also presented the code smell brain class in a similar way: complex classes that tend to accumulate an excessive amount of intelligence. The main difference is that a god class accesses directly many attributes of other classes.

God class, large class and brain class have a similar concept (Lanza and Marinescu 2005; Mäntylä and Lassenius 2006a). In our work, we consider the general idea of these smells. We adopt the term god class to refer to this general idea, indistinctly. Schumacher et al. (2010) well capture the essence of the concept in a set of support questions, which we adopt in our experiments:

- Does the class have more than one responsibility?
- Does the class have functionality that would fit better into other classes?
  - By looking at the methods, could one ask: "Is this the class' job?"
- Do you have problems summarizing the class' responsibility in one sentence?
- Would splitting up the class improve the overall design?

2.2 Empirical studies on code smell

As the use of the code smell concept has become widespread, empirical studies have been presented to help understanding its effect. In Section 2.2.1, we cover the use of tools for code smell detection, since tool support is one of the independent variables in our experiments. In Section 2.2.2, we present studies addressing the human role on smell detection: our family of experiments is part of this scenario. To the best of our knowledge, this is the first study analyzing such extensive set of independent variables with respect to code smell detection.

2.2.1 Tool assessment

There are available tools for code smell detection, such as JDeodorant[1] and inCode[2]. The tools are based on metrics and thresholds. Few studies on tool support for smell detection consider the "inherent uncertainty of the detection" (Khomh et al. 2009; Moha et al. 2010).

They evidence the necessity of works on tool assessment considering the human role on smell detection.

In this direction, software visualization has emerged as an alternative to address smell detection (Simon et al. 2001; Parnin et al. 2008; Murphy-Hill and Black 2010; Carneiro et al. 2010). Software visualization employs visual paradigms (visual resources, graphic design or animation) to facilitate both the human understanding and effective use of software (Price et al. 1998). Software visualization tools are also based on metrics. However, visual resources combined with metrics may help humans to identify design problems.

In some experiments presented in this paper, we use software visualization to assist code smell detection. In the following, we briefly address some research works that support our approach.

Murphy-Hill and Black (2010) presented a visualization tool implemented as an Eclipse plug-in. The tool is composed of sectors in a semicircle on the right-hand side of the editor panel, called petals: each petal corresponds to a smell. They performed a controlled experiment with 12 participants (six programmers and six students) to evaluate the tool. Their main findings were: i) programmers identify more smells using the tool than not using the tool; ii) smells are subjective; and iii) the tool helps in deciding what is and what is not a code smell.

Carneiro et al. (2010) presented the SourceMiner tool, a multi-perspective visualization environment implemented as an Eclipse plug-in. SourceMiner has visualizations addressing software characteristics such as inheritance and coupling. The visualizations also portray previously mapped concerns of the software. The authors performed an exploratory study using the concern mapping multi-perspective approach to identify code smells. Two main findings are presented. First, the concern visualizations provided useful support to identify God Class and Divergent Change smells. Second, the authors were able to identify strategies for smell detection, supported by the multiple concern views.

2.2.2 Role of human beings

This type of study explores factors impacting the human perception of code smells. As previously discussed, our family of experiments is part of this scenario. It is important to note that most of these studies aim to identify factors related to the human perception of smells. The identification of these factors is not our aim. We look for establishing a cause-effect relationship between some specific factors and the human perception of smells from our family of controlled experiments. To the best our knowledge, there are few studies exploring this cause-effect relationship. Due to this, this section presents the main ones we found in the literature about human perception of code smells, independent of their aims.

Palomba et al. (2014) investigated if what developers believe to be a problem is actually a problem. They adopted survey as the experimental method, considering different types of participants: i) graduate students; ii) industrial developers and; iii) developers of the systems themselves. Their findings show that some smells are generally not perceived as a design problem: *class data should be private*, *middle man*, *long parameter list*, *lazy class*, and *inappropriate intimacy*. They also noted "instances of a bad smell may or may not represent a problem based on the 'intensity' of the problem". Another finding was that developers consider large/complex source code as an important threat. Finally, they noted that developer experience and system's knowledge play an important role on smell

detection. Thus, they focus on the human role from a survey and they consider one of the variables that we address, which is the developer experience.

Moonen and Yamashita (2012) analyzed the perception of participants on the maintainability of the systems and related it back to code smells. They adopted case study as the experimental method, conducting interviews in an in vivo setting. The findings were added to another empirical study performed with experts and using the same set of systems (Anda 2007). They identified thirteen factors, such as *appropriate technical platform* and *simplicity*, affecting maintainability according to the participants. They found that eight of the factors affecting maintainability are addressable by current code smell definitions. However, in most cases these code smells would need to be complemented with alternative approaches, such as semantic analysis and manual inspection, in order to help on identification of maintainability factors. Thus, they also consider factors impacting in the human perception of design problems, which is similar to our objectives.

Mäntylä (2005) presented results of two studies addressing agreement in smell detection and factors to explain it. He adopted controlled experiments as the experimental method. He found high agreement for simple code smells - *long method* and *long parameter list* - and weaker agreement concerning the *feature envy* code smell and refactoring decisions. He found good correlation between the metric *lines of code* and the smell *long method*; and the metric *number of parameters* and the smell *long parameter list*. However, metrics were not useful to explain evaluation of *feature envy* and refactoring decisions. Finally, he found low correlation between refactoring decisions and demographic variables, such as developer's years of experience. Developer experience is one of the factors that we are addressing in our experiments.

Mäntylä and Lassenius (2006b) investigated why and when people think a code needs refactoring. They analyzed one of the experiments presented in Mäntylä (2005) to investigate what issues in code define the refactoring decisions. They refer to these issues as drivers. They adopted survey as the experimental method, applying a questionnaire to understand refactoring decisions. The most important driver was the size of a method. One of their important findings was that there was a conflict of opinions among the participants with respect both to the assessed internal quality of the methods and the need to refactor them. They also found that some drivers are difficult or impossible to be detected automatically, and some code smells are better detected by experienced participants than automatically. In our case, we considered some drivers identified by Mäntylä and Lassenius (2006b), such as software size and developer's experience.

Schumacher et al. (2010) built on and extended Mäntylä and Lassenius (2006b). They investigated the way software developer professionals detect the *god class* code smell. Then, they compared these results to automatic classification. They adopted controlled experiment as the experimental method, carrying it out in a professional environment. Their main findings were: (1) there was low agreement among participants detecting god class and (2) *misplaced method* was the strongest driver for *god class* detection. Related to the evaluation of automatic detection, their main findings were: (1) an automated metric-based pre-selection decreases the effort needed for manual code inspections and (2) automatic detection followed by manual review increases the overall confidence. Despite differences in the analysis procedure, Schumacher et al. (2010) investigated human perception of god class, as we perform.

Mäntylä and Lassenius (2006a) and Mäntylä et al. (2004), investigated agreement and the impact of demographic data, such as experience of developers, on smell detection by humans. Moreover, they compared results of human evaluation with metric-based heuristics. They adopted survey as the experimental method. In one of the findings the authors declare: "*the use of smells for code evaluation purposes is hard due to conflicting perceptions of different evaluators*". They also found that, some demographic variables, such as developer experience, partly explain the variation. Related to the correlation of human evaluation and the metric-based heuristics, they analyzed only four smells: *large class*, *long method*, *long parameter list*, and *duplicate code*. For *long method* and *long parameter list*, human evaluation correlated well with the metrics. For *large class* and *duplicate code*, they did not perceive the correlation as well. From the drivers identified by Mäntylä and Lassenius (2006a) and Mäntylä et al. (2004), we detach the developer's experience as one of the factors we are addressing.

3 The family of experiments

In this section, we present our family of controlled experiments, which is called *Finding God Class* (*FinG*). FinG is proposed to explore a set of aspects related to god class detection. As discussed in the Section 1, we used FinG family to investigate effort, decision drivers and strategies adopted by participants during the detection of god classes. Readers interested in them should refer to the following papers (Santos et al. 2013, 2014; Santos and Mendonça 2014, 2015). This paper does not discuss these issues. It investigates different factors from different data and strategy of analysis.

Currently, FinG has four controlled experiments. We named FinG 1, the first experiment of FinG; FinG 2, the second experiment; and so on. Each experiment asks participants to detect god classes in different pieces of source code. FinG was iteratively defined as proposed by Mendonça et al. (2008): each experiment gave us insights to the next experimental setup.

For sake of completeness, we detail each experiment of FinG following the guidelines proposed by Jedlitschka et al. (2008). In Section 3.1, we present the experimental unit, motivation and reward. In Section 3.2, we present the tools used in the family of experiments. In Section 3.3, we present the forms and docs used to collect the data from the participants. In Section 3.4, we present the software objects analyzed by the participants. In Section 3.5, we present the task performed by the participants. In Sections 3.6, we present the design for each experiment in the family. In Section 3.7, we present the process we adopted in order to define our oracle. Finally, in Sections 3.8 and 3.9, we present the execution and deviations occurred during the realization of the experiments.

3.1 Experimental unit, motivation and reward

All experiments involved undergraduate or graduate students (including professionals) from the Federal University of Bahia (UFBA), in Brazil.

FinG 1. FinG 1 involved 11 junior (3rd year) undergraduate Computer Science students. All students were enrolled in the Software Quality course offered in the first semester of 2012. This is an optional subject of the Computer Science undergrad program, in which design quality and smells are addressed. The course was considered appropriate for the experiment, both because it was focused in related subjects and it was not

mandatory, which means that most students enrolled on it were interested in the subject. Furthermore, participation in the experiment was voluntary.

FinG 2, FinG 3 and FinG 4. The other three experiments involved graduate Computer Science students. All students were enrolled in the Experimental Software Engineering course offered in the second semester of 2012 (FinG 2), first semester of 2013 (FinG 3) and second semester of 2013 (FinG 4). There were 25 participants in FinG 2, 25 in FinG 3 and 17 in FinG 4. The participation in the experiments was mandatory. The participants received grades for their participation, but not for their performance.

From the analysis of the characterization form (we will discuss the forms later on), we noted that, in FinG 2, all participants were, or had been, software developer professionals. In FinG 3, only four participants were not software developer professionals. And, in FinG 4, only two participants were not professionals software developers. We considered the data of the six participants without professional experience in software development because their accuracy values, according to our oracle, were higher than the average results of the other participants.

3.2 Tools

FinG 1, FinG 2 and FinG 3. In these experiments, we adopted two main software tools[3]: Eclipse, a well-known software IDE; and SourceMiner, an Eclipse plug-in that provides visual resources to enhance software comprehension activities.

SourceMiner enhances the comprehension of relevant software attributes on god class detection, in comparison with the traditional Eclipse IDE setup (Carneiro et al. 2010; Carneiro and Mendonça 2013, 2014). We detach the observation of size and coupling relations among software classes. In order to observe these relations, developers using Eclipse typically analyze the package-class-method structure of programs, by collapsing and expanding the nodes of the vertical tree into the Package Explorer view (Fig. 1a). Then, they open and read classes in the Source Code Editor view (Fig. 1b) to identify size and coupling attributes. The two views present problems. Kersten and Murphy (2006) discuss limitations related to the use of Package Explorer for developers navigating between

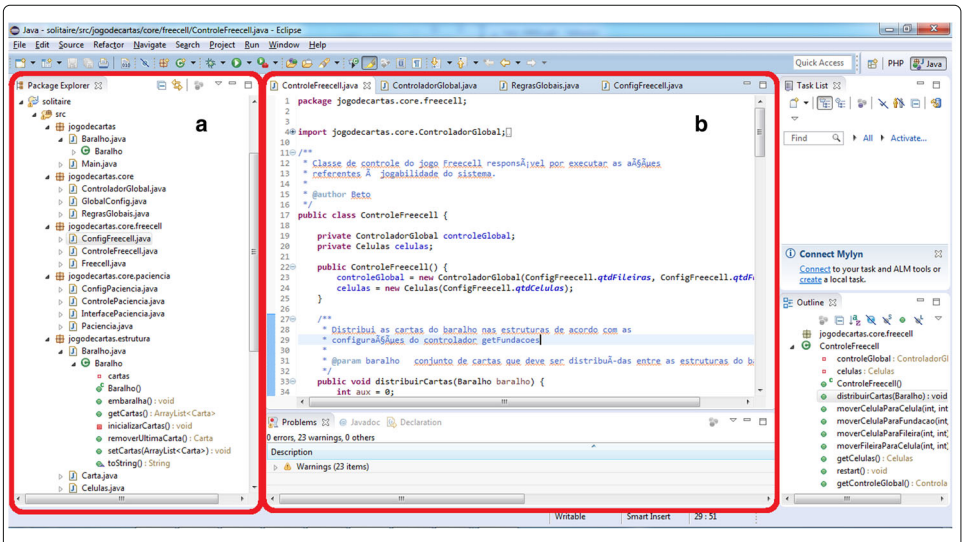

Fig. 1 Eclipse IDE: (**a**) the package explorer view and (**b**) the source code editor view

software artifacts in different modules; and de Alwis and Murphy (2006) discuss difficulties on software comprehension by reading lines of code. Using the SourceMiner, software structure and size and coupling attributes are observed through the graphical resources.

The SourceMiner has five visualizations, divided into two groups. The first group is composed of three software coupling visualizations. They show different types of dependency among entities, like direct access to attributes or method calling, for instance. They also show the direction of the coupling. The coupling views are based on radial graphs (Fig. 2a), relationship matrix (Fig. 2b), and tabular view (Fig. 2c). As an example for comparison with the use of the traditional Eclipse IDE, lets look the view based on radial graphs (Fig. 2a). Circles represent classes, and arrows represent dependencies among classes. A tool-tip, informing data about the class, appears when the cursor is positioned over the class. Moreover, double click opens the code of the class on the Source Code Editor. It is possible to observe coupling attribute for all classes looking the radial graph view (and the other coupling views of SourceMiner). The tool also has interaction resources enhancing comprehension of the code design (Carneiro et al. 2010; Carneiro and Mendonça 2014). Using the traditional Eclipse IDE setup, it is necessary to read the source code of each class in order to identify coupling.

The second group is composed of two hierarchical visualizations. The Treemap view (Fig. 2d) shows the hierarchy of package-class-method of the software. A Treemap is a hierarchical 2D visualization that maps a tree structure into a set of nested rectangles (Johnson and Shneiderman 1991). In SourceMiner, rectangles representing methods of the same class are drawn together inside the rectangle of the class. Likewise, the rectangles of the classes that belong to the same package are drawn together inside the rectangle of the package. Lines of code (LOC) or cyclomatic complexity are associated to the area and colors of the rectangles. Thus, it is possible to observe classes with higher LOC value easier than opening and reading the classes, such as using the traditional Eclipse IDE. The Polymetric view (Fig. 2e) shows the hierarchy between classes and interfaces. A Polymetric view uses a forest of rectangles to represent the inheritance trees formed by classes and interfaces in a software system (Lanza and Ducasse 2003). In SourceMiner,

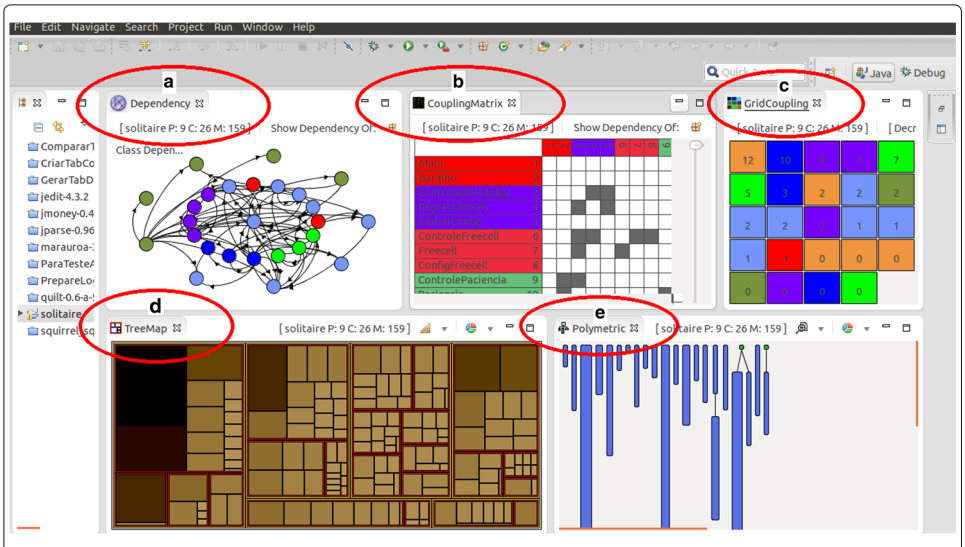

Fig. 2 SourceMiner views

rectangles are linked by edges representing the inheritance relationship between them. The length and width of the rectangles are used to represent the size and number of methods of a class.

FinG 4. In FinG 4, we adopted only the Eclipse IDE tool. We did not adopt the software visualization infrastructure.

3.3 Forms and docs

Five forms were used in total. During the training, participants filled a Consent and a Characterization form. The Consent form was adopted because of ethic issues: the participants had to consent publishing results of the experiment. The characterization form was mainly adopted to group the participants in our analyses.

Except for FinG 4, where we did not adopt software visualization, the participants also received a SourceMiner exercise guide. The adoption of this form supports the use of the tool by the participants.

During the experiment itself, they filled in an Answer form where participants had to fill in: i) initial and end time of the each smell detection task, and ii) identify each candidate god class. This was the data form.

At the end of the experiments, the participants filled a Feedback form. On it, we asked the participants to classify the training and the level of difficulty performing the detection tasks. It was also possible to write down suggestions and observations about the experiment. We adopted a Feedback form to better learn about the experiments.

Besides these forms, we used two other documents, during experiments themselves. One of them presented an overview of the software artifacts adopted in each experiment. The other was a Support Question guide used to steer the participants in the search for god classes. The questions were the same ones used by Schumacher et al. (2010). We presented them in the very end of Section 2.1.

3.4 Software artifacts

FinG 1 and FinG 2. Six programs were used in these experiments. All of them implement simple applications or games in Java. We chose Java language because the experience of the participants and oracle researchers. Chess, Tic Tac Toe, Monopoly and Tetris implement well known games. Solitaire-Freecell (Solitaire) is a framework for card games with Solitaire and Freecell. Jackut is a very simple social network application. Table 1 characterizes the programs in terms of the number of packages, classes and LOC.

FinG 3 and FinG 4. Four programs were used in these experiments. We adopted more complex software than in FinG 1 and FinG 2. The software were also implemented in Java: i) Quilt is a software development tool that measures test coverage; ii) JMoney is a personal finance (accounting) manager; iii) jParse is an Eclipse plugin in which you can parse XML returned from an Ajax request; and iv) SQuirrel SQL Client is a graphical Java program that allows one to view the structure of a JDBC compliant database, browse the

Table 1 Software objects for FinG 1 and FinG 2 experiments

Software	Chess	Jackut	Tic Tac Toe	Monopoly	Solitaire	Tetris
Packages	5	8	2	3	6	4
Classes	15	19	5	10	23	16
LOC	1426	978	616	2682	1758	993

Table 2 Software objects for FinG 3 and FinG 4 experiments

Software	Quilt	JMoney	Squirrel	jParse
Packages	20	4	3	4
Classes	104	79	73	69
LOC	8020	8197	6944	24796

data in the tables, and issue SQL commands, among other functions. Table 2 characterizes the programs in terms of number of packages, classes and LOC.

3.5 Task

FinG 1 and FinG 2. In these experiments, we asked participants to detect god classes in the software. We provided only a set of Support Questions as a guide. Each participant was free to use her/his own strategy to do the task.

FinG 3 and FinG 4. In these experiments, as in FinG 1/FinG 2, we gave the software and the Support Questions as a guide. However, we did not ask participants to detect god classes on all classes of the software. We selected 12 candidate classes in each software application, because the analysis of the whole software, during the experiments, was impractical. The 12 classes chosen are those with the highest LOC. This number is similar on average to the total number of classes in the small programs used in FinG 1/FinG 2. Moreover, this approach did not impact in the evaluation of the software size variable. Considering the classes observed by the participants of FinG 1 and FinG 2, the LOC value average is 93. Considering the classes observed by the participants of FinG 3 and FinG 4, the LOC value average is 684. The participants were free to define their process for detecting god classes.

3.6 Design

We consider the design of the experiments as the laboratory settings and the distribution of participants among the workstations. All experiments were carried out in a laboratory at UFBA. Participants had about three hours to carry out the task. Each participant worked at an independent workstation.

FinG 1 and FinG 2. In these experiments, the workstations were divided into two groups. In FinG 1, there were six participants in the Group 1 and five participants in the Group 2. In FinG 2, there were 13 participants in the Group 1 and 11 participants in the Group 2. At each workstation, we set up two Eclipse IDEs. We fitted the SourceMiner only for one of the Eclipse installations in the workstation. Each installation had three of the six programs in their workspace. The programs in the workspaces were rotated between the groups. Table 3 presents this design.

Table 3 Design of FinG 1 and FinG 2

Group	With SourceMiner	Without SourceMiner	Participants' ID (FinG 1)	Participants' ID (FinG 2)
1	Chess, Jackut and Solitaire	Monopoly, Tetris and Tic Tac Toe	14, 21, 32, 35, 42 and 44	1, 3, 4, 7, 9, 11, 13, 15, 17, 19 21, 23, 25 and 27
2	Monopoly, Tetris and Tic Tac Toe	Chess, Jackut and Solitaire	13, 15, 25, 31 and 41	5, 6, 8, 10 12, 14, 16, 18 20, 22 and 26

Table 4 Design of FinG 3

Group	With SourceMiner	Without SourceMiner	Participants' ID (FinG 3)
1	jParse and Quilt	JMoney and Squirrel	1, 2, 3, 6, 7, 10, 11, 12, 13, 21, 23, 24 and 25
2	JMoney and Squirrel	jParse and Quilt	9, 14, 15, 16, 17, 18, 19, 20, 27, 28, 29

FinG 3. The only difference in the design of FinG 3 compared with FinG 1 and FinG 2 was the number of programs in the workspace of each Eclipse IDE. In FinG 3, there was two medium sized, instead of three small sized, software in each Eclipse workspace. There were 13 participants in Group 1 and 11 participants in Group 2. Table 4 presents this design.

FinG 4. In FinG 4, we adopted the same medium sized software as FinG 3, but had only the basic setup of one Eclipse at each workspace. In this case, each participant always detected god classes without SourceMiner. In all, 17 participants performed the task for the four programs. Table 5 shows this design.

The participants were free to choose the order of the programs during the task. Thus, they also were free to choose the order of use of the visualization tool, when it was available.

3.7 The oracle

We adopted two different approaches to define the oracle in FinG 1-2 and FinG 3-4. We did this because the software used in FinG 1-2 are simpler than the software used in FinG 3-4. We detail the approaches as follows.

FinG 1 and FinG 2. The oracle of FinG 1 and FinG 2 was based on two experienced researchers from academia and industry. Both researchers had at least seven years working on different software companies and five years lecturing and researching in software engineering area. During the definition of the oracle answers, the researchers did not have any access to the answers of the participants. Each of the researchers did the god class detection task independently, based on their skills and experience. Then, they met to discuss their disagreements and to define the final oracle for the experiments. The two activities are important to bring confidence to our oracle process: 1) independent god class detection and 2) meetings for group discussion.

Table 6 shows the oracle of Fing 1 and FinG 2. First column shows the programs. Second column shows the number of classes observed by the researchers, which is the total number of classes of the programs. Third column shows the agreement among the researchers before their meeting (note that, there were few cases where the researchers disagreed on evaluation of candidate god classes). The last column shows the name of the candidate god classes as the oracle for FinG 1 and FinG 2.

Table 5 Design of FinG 4

Group	Without SourceMiner	Participants' ID (FinG 4)
1	jParse, Quilt, JMoney and Squirrel	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16 and 17

Table 6 The orcale of FinG 1 and FinG 2

Program	Total number of classes	Agreement before meeting	God class candidate
Tic Tac Toe	5	5 (100%)	-
Monopoly	10	9 (90%)	Jogo
Chess	15	14 (93%)	Chess
Tetris	16	16 (100%)	-
Jackut	19	19 (100%)	-
Solitaire	23	22 (96%)	-

FinG 3 and FinG 4. The oracle of FinG 3/FinG 4 was defined by three researchers, two of them with years of experience in academia and industry. The first researcher is one of the researchers of FinG 1 and FinG 2. The second researchers had at least 5 years working on different software companies and 4 years lecturing and researching in software engineering area. The third research was a graduate student that had five years working in software engineering area.

After identifying the twelve largest classes for each of the four programs, the following strategy was adopted to classify them:

1. To evaluate the name of the class, understanding its role on the software;
2. To evaluate the name of each method, understanding their role on the class;
3. For large or suspect methods, reading code;
4. If the method is considered out of scope (considering the role of the class), then to mark it as such;
5. If two methods are considered out of scope, then the class should be classified at least as a "maybe" god class (this threshold is defined to mark the class for further discussion among the oracle researchers);
6. If more than two methods are considered out of scope, the class is a "yes" god class;
7. If doubts remain, considers size and readability to determine if the class is a god class.

The experimenter, who was one of the oracle researchers, defined the god class detection strategy. It is important to highlight that the strategy above is presented to harmonize the perception of the oracle researchers, instead to be considered a definitive approach to detect god class. The main aspect of the oracle definition was the group discussion, as we present below.

Alone, the first researcher classified the 12 classes selected for each program as a sure or may be god class. The other two researchers were trained on the proposed strategy using an independent set of classes. Together, they investigated pairwise the 12 selected classes. After that, the three researchers met to compare and discuss their answers. During the meeting, the discussion among the researchers focused on how flexible they should be considering size and methods out of scope for the classes, such as the oracle researchers of FinG 1/FinG 2.

Table 7 shows as the oracle researchers agreed on the identification of the candidate (and not candidate) god classes. The table has the same structure as Table 6.

It is important to note that we did not use any automatic detection strategy because all heuristics are based on metrics and thresholds. Once we were interested in investigation of factors affecting the human perception of god classes, we did not provide any metrics

Table 7 The oracle of FinG 3 and FinG 4

Program	Total number of classes	Agreement before meeting	God class candidate
JMoney	12	10 (83%)	MainFrame
JParse	12	8 (67%)	Type
Squirrel	12	12 (100%)	-
Quilt	12	12 (100%)	-

values, avoiding to influence the participant answers. Moreover, we adopted the general concept of god class, which involves concepts of god class, brain class and large class, as discussed in Section2.1. Our analyses are not affected by the programs with no candidate god classes. We grouped the results for each investigated factor, considering the agreement among the participants and between the participants and the oracle (we detail our strategy of analysis in Section 4.3).

3.8 Execution

The standard experimental setup took four days, in a span of three weeks. Two days for training, one day for a pilot and one day for running the experiment itself. In the experiment day, all participants performed the task for all programs, using and not using the visualization tool, when it was permitted. The training was performed in two days due to the quantity of activities. We performed a motivational presentation in the first day of the training. In this presentation, we discussed the experimental software engineering scene and tied it with discussions about code smells because we noted that some participants had few knowledge on the topic. At the end of the first day, we asked the participants to fill out the Consent and Characterization forms. On the second day of training, we focused in the visualization tool. We performed the activity in a lab, introducing SourceMiner and running a practical exercise. The practical exercise focused on the use of the tool and how it could help on perception of differences on size, complexity and coupling among classes. The last activities of the exercise asked the participants to search for god classes using SourceMiner. For these activities, we did not influence the participants on their evaluation.

As discussed earlier, there were variations on the standard setup of the experiments. Table 8 shows the schedule and variations for each experiment. In the motivational presentation of FinG 1, we discussed shortly the concepts of code smell, large class, god class and brain class. We did this because the participants were undergraduate and had limited software development experience. For FinG 2, FinG 3 and FinG 4, we did not discuss

Table 8 The schedule and variations of the experiments

Day	Activity	FinG				Local	Time (Hour)
		1	2	3	4		
1	Motivational presentation	•	•	•	•	Classroom	1.0
	Code smell concept presentation	•				Classroom	0.25
2	Questionnaire after reading		•	•		Lab	0.25
	SourceMiner presentation	•	•	•		Lab	0.5
	SourceMiner exercise	•	•	•		Lab	2.0
	Training based on examples				•	Lab	2.5
3	Pilot	•	•	•	•	Lab	2.5
4	Execution	•	•	•	•	Lab	2.5

the god class or code smell concepts. We simply asked the participants to read parts of Fowler (1999) and Lanza and Marinescu (2005)'s books, instead. They received a document explaining which pages of the books they had to read. From Fowler's, they read about refactoring, smells and large classes. From Lanza and Marinescu's, they read about disharmony, god class, brain methods and brain classes.

In the following week, before starting their training on SourceMiner, participants of Fing 2 and FinG 3 were given a 15 min questionnaire. The questions (three in total) addressed concepts of refactoring, large class, god class and brain class. This requirement motivated the participants to read the training material more carefully and was later used to measure *developer knowledge* (one of our independent variables). FinG 4 training was different. There was no SourceMiner training. Instead, we performed a training based on examples. First, we showed examples of classes with different size, complexity and coupling, discussing why we consider them as candidate or not candidate to be a god class. We based on the process that we defined in Section 3.7. Then, we asked the participants to perform the same activity looking other classes and identifying if they were god class candidate or not. They answered in a WEB form that we created. In the final part of the day training, we (the experimenters and participants) discussed about the differences on the answers. The aim of this discussion was to align the personal thresholds and rigor on the evaluation of god classes in order to support us on the evaluation of the *developer training* factor (another of our independent variables).

3.9 Deviations and non-conformities

We discarded some data points in each of the first three experiments. FinG 1 started with 17 participants, but we used only 11 data points; FinG 2 started with 28 participants, but we used only 25; and FinG 3 started with 29, but we used only 25. For all cases, the results were discarded because, or the participants participated of the pilot, or they missed, at least, one of the experiment's activities. FinG 4 started with 17 participants, all completed the experiment.

Except for FinG4, we performed a pilot with two participants. In FinG 1, the pilot helped us to evaluate the use of the Answer form, in paper or electronic format, and to evaluate the time needed to complete the experiment. In FinG 2, the pilot did not indicate any problem, probably because its setup is very similar to FinG 1. The pilot was useful in FinG 3 because it helped us to adjust the Answer form, which was different from FinG 1 and FinG 2, and to evaluate the time, because we asked the participants to analyze 12 classes of four medium size programs (more complex that those used in FinG 1/FinG 2). Finally, the pilot in FinG 4 used two independent researchers. It helped us to evaluate and to perform some adjustments in the training on god class concept, which was based on examples.

4 Evaluation of five factors affecting god class detection

In this section, we discuss our approach to address five factors affecting the human perception on god class detection. We detail the analyzed factors, variations in the experimental setup and the analysis procedure.

Figure 3 summarizes our approach. We measured the human perception as the agreement among developers detecting god classes: low agreement indicating that the factor significantly impacts the human perception on god class detection. Thus, we defined "agreement" on god class detection as the dependent variable (the arrow on the right side

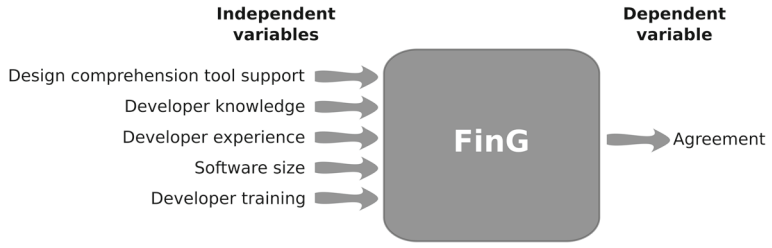

Fig. 3 The investigated factors

of the figure). We discuss our definitions of agreement later on. We evaluate the impact of the following factors as the independent variables of our experiments (the arrows on the left side of the figure): design comprehension tool support, developer knowledge and experience, software size, and developer training.

4.1 The choice of the investigated factors

In order to investigate the factors, we define them as the independent variables of our controlled experiments. Constraints on carrying out controlled experiments force us addressing a limited number of variables. The choice of input variables was based on insights gained after the analysis of other empirical works on the subject, or the analysis of the previous experiment in the FinG family. We based the choice of the variables on the context definitions proposed by Dybå et al. (2012). They define three dimensions for the discrete context, which is focused on variables directly affecting the behavior in software engineering. The dimensions are technical, social and environmental. In this paper, we address two types: technical (size of software and tools adopted) and social (knowledge, experience and training). As previously discussed, we did not find other study analyzing such extensive set of independent variables with respect to smell detection agreement. Below, we explain and motivate the choice of the variables in more detail:

- **Design comprehension tool support**. One of our first ideas to explore factors affecting smell detection was based on the nature of the concept. As the code smell concept is intrinsically related to the quality of design, we decided to investigate the impact of the overall comprehension of the design on human evaluation of god class, one of he most known smells. To do this, we considered the use of software visualization. We found some studies addressing software visualization tools and code smell detection (Simon et al. 2001; Van Emden and Moonen 2002; Parnin et al. 2008; Murphy-Hill and Black 2010), but their focus was on the evaluation of the tools. None of them focused on discussing how a better comprehension of the design (acquired by the use of visual resources) affects the human perception of a code smell.
- **Developer knowledge**. Dybå et al. (2012) highlighted individual skills when discussed the importance of context in experimentation. They consider individual skill as a variable that directly might influence behavior or moderate relationships between other variables. We investigated if extensive reading about god classes and code smell would impact on god class detection agreement. We did not find other studies addressing this variable.
- **Developer experience**. The experience is an important contextual aspect in the software engineering discipline (Höst et al. 2000; Carver et al. 2003; Höst et al. 2005).

An evidence of the importance of experience in smell detection is noted in Kreimer (2005)'s statement: "*depending on the perception and experience of the searching engineer, design flaws are interpreted in a different way*". Despite the importance, the topic is lightly addressed in code smell evaluations. Even Kreimer's work did not explore the impact of the experience on code smell detection. He proposed a method to find code smells.

- **Software size**. In FinG 1 and FinG 2, we looked at simple and familiar applications to minimize the effort of participants during the inspection of code. Our argument was that the human evaluation is affected by subjectivity for any type of software. We felt the need to assess if software size, an important factor, plays a role in smell detection agreement. FinG 3 was used to investigate this subject.
- **Developer training**. This variable emerged from the analysis of FinG1, FinG 2 and FinG 3. We found out that inconsistencies on code smell detection were not mitigated by the use of tool support, such as using metrics or visualization resources. We then hypothesized that, in order to mitigate subjectivity, it is important to teach people about the conceptualization of smells. We decided to use training, based on golden examples[4] of smells and group discussions, as one of the independent variables in our work.

4.2 Variations in the experimental setup

This section shows how we controlled the independent variables, varying some attributes of the experiments. The experimental variation in a family of controlled experiments, to expand and to evolve the knowledge on a topic, is proposed by Basili et al. (1999) and Juristo and Vegas (2009). Figures 4 and 5 show how we controlled the addressed variables from two approaches: i) in some cases, we investigated the variables independently for each experiment; ii) in the other cases, the investigation was based on the comparison among the experiments' results.

In Fig. 4, we show variables that were investigated independently for each experiment. In these cases, we used two experiments to strength our analysis of each variable. In other words, we performed our analysis based on two different and independent set of evidences. We chose the two experiments with the closest experimental setup in order to perform these analyses. This was the case of the *design comprehension tool support* and *developer knowledge* variables:

- **Design comprehension tool support**. Participants detected god classes with and without the use of software visualization tool support, in FinG 1 and FinG 2 (Fig. 4, left side). In these experiments, we evaluate the results of the agreement in both cases with and without visualization.

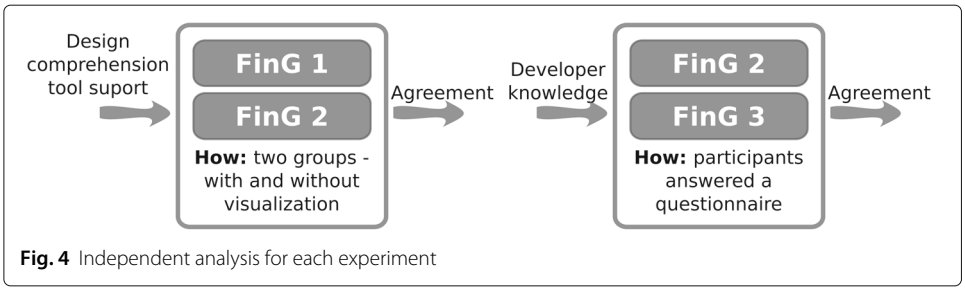

Fig. 4 Independent analysis for each experiment

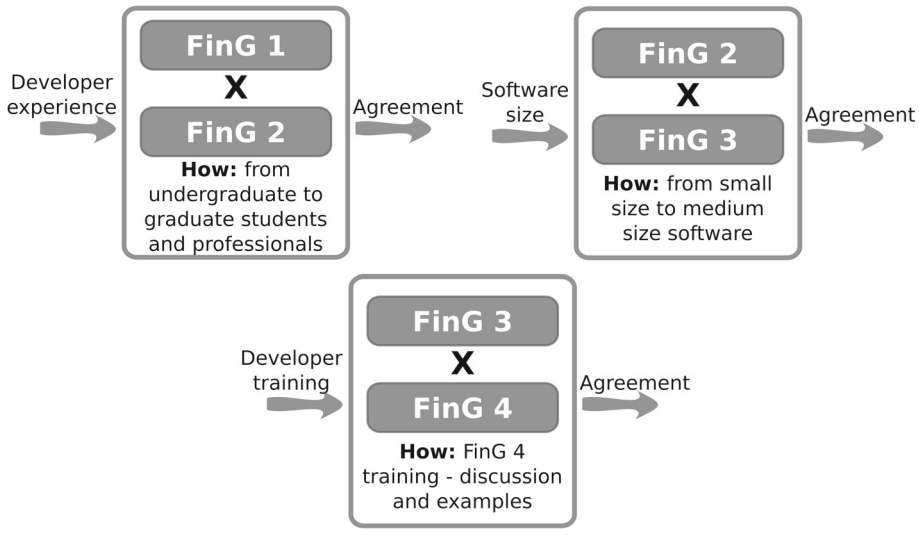

Fig. 5 One experiment against the other

- **Developer knowledge**. During the training, participants received a questionnaire, where they answered conceptual questions about code smell and god class, in FinG 2 and FinG 3 (Fig. 4, right side). We evaluate the agreement among participants with "good" and "bad" knowledge rating, independently, for each experiment.

In Fig. 5, we show variables that were investigated from the comparison between the experiments' results. In these cases, we considered results of one experiment against the other experiment in order to analyze if the experimental setup changing affected the participants answers. It was the case of the *developer experience*, *software size* and *developer training* variables:

- **Developer experience**.
  The main difference on the experimental setup of FinG 1 and FinG 2 was the experience of the participants (Fig. 5, top left side). While FinG 1 was carried out with undergraduate students, FinG 2 only had graduate students and professionals. We then compared results of FinG 1 against results of FinG 2 to investigate the impact of experience on smell detection agreement.
  It is important to highlight that there is another difference among FinG 1 and FinG 2, which is the training. We present it in Section 3.8 and summarize in Table 9. In FinG 1, the training was based on a presentation, while in FinG 2 the training was based on book reading. We consider that this difference does not impact our analysis. We argue that in both FinG 1 and FinG 2 cases, the training approaches just shallowly exposed the participants to a quite simple and intuitive concept, which is god class. The reasons about why we switched the training are discussed in Section 7 (Threats to validity).
  We disregarded the results of FinG 3 and FinG 4 in this analysis because, despite similar participants' experience, the inspected software were different.
- **Software size**. The main difference on the experimental setup of FinG 2 and FinG 3 was the size of the analyzed software (Fig. 5, top right side). We considered line of code (LOC) as the measure of software size. FinG 2 was carried out with six small

Table 9 Summary of experimental setup of FinG's experiments

Experiment	Participants	Software	Design	Training
FinG 1	Undergraduate students	Six small software artifacts	Two groups: with and without visualization tools	Presentation
FinG 2	Graduate students/ professionals	Six small software artifacts	Two groups: with and without visualization tools	Reading books and questionnaire
FinG 3	Graduate students/ professionals	Four medium software	Two groups: with and without visualization tools	Reading books and questionnaire
FinG 4	Graduate students/ professionals	Four medium software	One group: without visualization tools	Reading and training based on examples

programs, while FinG 3 used four medium size programs. Thus, we compared the results of FinG 2 against the results of FinG 3 to investigate the impact of software size on smell detection agreement.

We highlight that there is another difference among FinG 2 and FinG 3, which is related to the task performed by the participants (see Section 3.5). The participants of FinG 2 were asked to detect god classes inspecting all software, while the participants of FinG 3 were asked to detect god classes for specific classes. We consider that this difference does not impact our analysis, because the participants inspected a quite similar number of classes in both FinG 2 and FinG 3 experiments (around 12). The reasons for task changing are discussed in Section 7 (Threats to validity).

We disregarded results of FinG 1 because the participants were undergraduate students. In the same way, we disregarded results of FinG 4 in this analysis because its training was significantly modified as it is one of the variables we investigate.

- **Developer training**. The main difference on the experimental setup between FinG 3 and FinG 4 was the training of the participants (Fig. 5, bottom). The participants of FinG 3 just read a set of training material and answered an evaluation questionnaire, while the Participants of FinG 4 read the training material, had a training lecture with examples and group discussion about smells. We then compared results of FinG 3 against results of FinG 4 to investigate the impact of training on smell detection agreement.

  We highlight that there is another difference between FinG 3 and FinG 4, which is related to the use a support visualization tool (see Section 3.2 and Table 9). The participants of FinG 3 used a visualization tool, while the participants of FinG 4 did not use a visualization tool. We consider that the absence of a visualization tool in FinG 4 does not impact our analysis because this was one of our findings investigating the independent variable "design comprehension tool support". This analysis is presented in Section 5.1.3.

  We disregarded results of FinG 1 and FinG 2 because the software used in these experiments were different. We detail type of software for each experiment in Section 3.4.

Table 9 summarizes the variations between the experiments. The first column contains the name of the experiments, while the other columns summarize the participants, the software, the experimental design and the training for each FinG experiment.

4.3 Analysis procedure

Our analysis considered two perspectives: agreement among the participants and agreement between each participant and the oracle.

Agreement among the participants. We adopted the Finn coefficient (Finn 1970), as opposed to the Kappa coefficient (Cohen 1960; Fleiss and et al 1971), used in other works addressing code smells (Schumacher et al. 2010; Fontana et al. 2012; Yamashita and Counsell 2013). These studies adopted Cohen's Kappa coefficient, which is an index of inter-rater reliability used to measure level of agreement among two raters. Fontana et al. (2012) also adopted Fleiss' Kappa, which is a coefficient used to measure level of agreement for more than two raters. As the Fleiss' Kappa, Finn coefficient is used for more than two raters.

We adopted Finn coefficient because of problems identified in Kappa coefficient, by data analysis researchers (Whitehurst 1984; Feinstein and Cicchetti 1990; Gwet 2002; Powers 2012). The Kappa test is done in two phases. First, an agreement rate is calculated and, then, this value is used to calculate the coefficient. Feinstein and Cicchetti (1990) show that one can have high agreement rate and low values of the Kappa coefficient when the variance on values of raters is low. We noted this situation in the work of Schumacher et al. (2010). The Finn coefficient is recommended when variance between raters is low (Finn 1970). Whitehurst (1984) suggests Finn as an alternative to problems with Kappa, and affirms that it is the most reasonable index for agreement.

Our analyses were based on the evolution of the agreement level between the different groups analyzed (with and without visualization support, more and less experienced participants, small and medium software size, knowledge level, and training). We adopted classification levels to make it easier the comparison between the agreement values. Like Schumacher et al. (2010) and Zhang et al. (2011), we used the agreement classification levels defined by Landis and Koch (1977). The classification is as follows: slight, for values between 0.00 and 0.20; fair (between 0.21 and 0.40); moderate (between 0.41 and 0.60); substantial (between 0.61 and 0.80); and almost perfect (between 0.81 and 1.00) agreement level.

A problem occurs with the use of agreement coefficients when there is a high number of items that have a sure negative classification. In these cases, the agreement coefficient tends to be very high, hiding disagreements on the items with possible positive classifications. This occurs in FinG 1 and FinG 2, where most of the classes are small and clearly are not god classes. To mitigate this problem, we considered that the total number of classes to be used in the agreement test was twice the total number of god class candidates. We defined a candidate god class as class chosen by, at least, one participant. If this number was higher than the total number of classes of the software under analysis, we considered the total number of classes of the software. This problem did not occur in FinG 3 and FinG 4, because we selected the biggest 12 classes of medium sized software, most of them good god class candidates.

Agreement between the participants and the oracle. To evaluate participant success with respect to the oracle, we adopted an accuracy measure. The confusion matrix shown in Table 10 captures the idea of our analysis. The top columns values represent candidate god classes selected by the participants. The left column values represent candidate god classes indicated by oracle. True positives represent the case where the participant and oracle agree on a candidate god class. False positives occur when participants marked a

Table 10 Confusion matrix

		Predicted (participant)	
		God class	No god class
Actual (oracle)	God class	True positive (TP)	False negative (FN)
	No god class	False positive (FP)	True negative (TN)

god class in disagreement with the oracle. False negatives occur when the participant did not mark a god class in disagreement with the oracle. Lastly, true negatives are the cases where the participant and the oracle agreed on a negative.

The accuracy measure captures all these cases. The formula is given below.

$$accuracy = \frac{number\ of\ :\ TP + TN}{number\ of\ :\ TP + FP + FN + TN} \quad (1)$$

After calculate accuracy values, we statically compared the different groups analyzed (with and without visualization support, more and less experienced participants, small and medium software size, knowledge level, and training). We adopted the Shapiro-Wilk normality test, which is considered the most powerful normality test by some authors (Kitchenham et al. 2017). For all cases, there was, at least, a group of the samples in which the distribution was not normal. Due to this, we adopted the Mann-Whitney, a non-parametric alternative to t-test, with a 0.05 p-value, to statistically test our hypotheses, which we present in the next section.

We also evaluated our results in terms of magnitude, testing the effect size measure. We adopted the non-parametric Cliff's Delta test (Cliff 1996) to evaluate the effect size. Cliff's Delta test is also recommended by Kitchenham et al. (2017). In order to assess the magnitude on effect size, we used the classification presented by Romano et al. (2006). The classification is as follows: negligible, for $|d| < 0.147$; small ($|d| < 0.33$); medium ($|d| < 0.474$); and large, otherwise. We considered Cliff's Delta values on small (or higher) classification as evidences of differences on the sample values. We did this because our samples had a small number of data points, which is common for controlled experiments in software engineering involving humans as participant subjects.

4.4 Hypotheses

Our hypotheses were based on the discussion about the independent variables, presented in Section 4.1.

Design comprehension tool support. Our hypothesis is: *developers detecting god class using visual resources to enhance their overall comprehension of the design agree and succeed (agreeing with oracle) more than developers not using any visual resource*. This hypothesis is based on the fact code smells represent "bad" design decisions and the visualization tool we adopted enhances the comprehension of the design.

Developer knowledge. The hypothesis is: *developers who have better knowledge on code smell agree and succeed more than the others*. We used a questionnaire, where participants answered simple questions about concepts related to god class to group the participants: those with "bad" and those with "good" knowledge on code smell concepts.

Developer experience. The hypothesis is: *more experienced developers agree and succeed more than less experienced developers*. We used the participants of FinG 1 and FinG 2, who had different experience profile, to test this hypothesis.

Software size. Our hypothesis is: *developers agree and succeed more if they are looking at simpler software*. We compared the results of FinG 2 and FinG 3, which had similar design and participant profile, but different software size, to test this hypothesis.

Developer Training. The hypothesis is: *better trained developers agree and succeed more than the others*. We compared the results of FinG 3 and FinG 4, which had similar participant profile and software size, but different training procedures, to test this hypothesis. Here it is important to observe that FinG 4 did not use the visualization tool in any treatment, while FinG 3 did that for half of its trials. This raises the concern that *Design Comprehension Tool Support*, an important confounding factor, was not considered in the analysis of the *Developer Training* variable. However, as will be presented in Section 5.1, FinG 1 and FinG 2 experiments showed that *Design Comprehension Tool Support* does not affect the agreement results.

5 Results

In this section, we present the results of our analyses for all independent variables we address.

5.1 Design comprehension tool support

To analyze how design comprehension impacts agreement on god class detection, we considered FinG 1 and FinG 2 experiments, independently. We grouped the participants using and not using visualization, because we consider that the visual resources of SourceMiner enhance overall comprehension of the program design (Carneiro et al. 2010; Carneiro and Mendonça 2013).

5.1.1 Agreement among the participants

Table 11 contains the agreement coefficient values for each program, considering both *with* and *without* visualization cases in FinG 1. The visualization column indicates if the experiment was realized with or without visualization tools. The other columns are the total number of classes (#class) considered in the agreement test, the number of participants (#part), the Finn coefficient value (Finn), the p-value and the agreement level.

In order to analyze the evolution of the agreement coefficient, from the case without visualization to the case with visualization, we used classification defined by Landis and Koch, as explained in Section 4.3.

In the second line (Table 11), the level of agreement related to Tetris program was "moderate" in both without and with visualization cases. The evolution value was filled with the term "SAME", indicating there was no evolution in the agreement level. For the Jackut program, the evolution value indicates the agreement level increased when participants used visual support.

Some agreement values are not filled in the table because the p-value is out of the significance range (> 0.05). This indicates that we cannot be confident about these coefficient values and these cases were removed from the analysis. Overall, considering the FinG 1 experiment, the agreement among the participants using visualization was better in one case. Three cases were not considered. And, the agreement level was the same in the other two cases.

Table 11 Finn agreement test for FinG 1 with and without visualization tool

Program	Visualization	#class	#part	Finn	*p*-value	Agreement level
Monopoly	Without	4	6	0.567	0.0135	Moderate
	With	2	5	0.4	0.221	-
	From without visualization to with visualization evolution					-
Tetris	Without	4	6	0.567	0.0135	Moderate
	With	6	5	0.467	0.0307	Moderate
	From without visualization to with visualization evolution					SAME
Tic Tac Toe	Without	4	6	0.567	0.0135	Moderate
	With	2	5	0.6	0.0788	-
	From without visualization to with visualization evolution					-
Chess	Without	6	5	0.467	0.0307	Moderate
	With	6	6	0.6	0.0014	Moderate
	From without visualization to with visualization evolution					SAME
Jackut	Without	6	4	0.5	0.0403	Moderate
	With	6	6	0.667	0.000226	Substantial
	From without visualization to with visualization evolution					INCR
Solitaire	Without	4	5	0.4	0.113	-
	With	8	6	0.617	0.000147	Substantial
	From without visualization to with visualization evolution					-

Table 12 presents the same structure of Table 11, but with the results of the FinG 2 experiment. In FinG 2, the agreement was worse when participants used the visual resources in two cases, and it was the same in the other four cases.

For the Jackut program, the result was the opposite of the result found for FinG 1 (Table 11). In FinG 1, the agreement level increased when the participants used visual support, while in FinG 2, the agreement level decreased.

Table 12 Finn agreement test for FinG 2 with and without visualization tool

Program	Visualization	#class	#part	Finn	*p*-value	Agreement level
Monopoly	Without	8	14	0.717	3.68e-14	Substantial
	With	6	11	0.636	1.44e-06	Substantial
	From without visualization to with visualization evolution					SAME
Tetris	Without	6	14	0.758	5.19e-13	Substantial
	With	8	11	0.636	2.94e-08	Substantial
	From without visualization to with visualization evolution					SAME
Tic Tac Toe	Without	4	13	0.667	3.73e-06	Substantial
	With	5	11	0.68	1.17e-06	Substantial
	From without visualization to with visualization evolution					SAME
Chess	Without	8	11	0.755	4.2e-13	Substantial
	With	8	14	0.739	1.67e-15	Substantial
	From without visualization to with visualization evolution					SAME
Jackut	Without	6	11	0.612	4.98e-06	Substantial
	With	4	14	0.516	0.000604	Moderate
	From without visualization to with visualization evolution					DECR
Solitaire	Without	6	11	0.745	7.38e-10	Substantial
	With	6	14	0.579	1.54e-06	Moderate
	From without visualization to with visualization evolution					DECR

5.1.2 Agreement between the participants and the oracle

As discussed in Section 4.3, we calculated the accuracy of the participants' answers, considering our oracle. We expected higher accuracy values for the cases where participants had a better comprehension of the design. Based on discussion about our hypothesis in Section 4.4, we defined the null hypothesis as:

- *H0: there is no difference of accuracy values between the cases where participants were using and not using visualization tool*

Table 13 shows the statistical tests. In the first line, we considered results for FinG 1. In the second line, we considered results for FinG 2. In both cases, we could not reject the null hypothesis using the Mann-Whitney non-parametric test (notice that the distribution is not normal according to the Shapiro-Wilk test). The final columns of the table show the Cliff's Delta effect size. The magnitude on the effect size, for both experiments, was negligible.

5.1.3 Analysis

We found *a design comprehension tool support does not improve agreement on god class detection*. The following highlights the evidences. First, the level of agreement among the participants for both using and not using visualization groups was the same for the most cases. Second, the hypothesis test and effect size were not conclusive. Besides, there were some contradictory results. In some cases, participants of FinG 1 had better results using visualization (case of the Jackut program in Table 11), and participants of FinG 2 had better results not using visualization (cases of the Jackut and Solitaire programs in Table 12). We have to consider the experience of the participants as a potential confounding factor in this analysis.

We conjecture that god class detection is more related to personal conceptualization than to design comprehension, and that technical support does not affect the conceptualization of the smell. We believe this is transversal to other kinds of technical support. Metric-based tools, for example, are dependent on heuristics and value thresholds, which are dependent on human definition anyway. Fontana et al. (2012) discuss some inconsistencies in heuristics using metrics-based tools.

5.2 Developer experience

To evaluate how experience impacts agreement on god class detection, we compared results of FinG 1 against results of FinG 2. The experiments had similar setup and the main difference between them was the participants' profile. We captured experience based on two questions of the Characterization form. The first question asked how long the participant worked professionally (even as a researcher) on software development. The second

Table 13 Hypothesis test for analysis of design comprehension tool support, considering FinG 1 and FinG 2

FinG	Shapiro-Wilk				Mann-Whitney		Cliff's Delta effect size	
	Without vis		With vis					
	W	*p*-value	W	*p*-value	W	*p*-value	Delta	Magnitude
1	0.8045	4.953e-05	0.7688	8.641e-06	478	0.4884	0.09	Negligible
2	0.7723	2.34e-09	0.7867	4.619e-09	3070	0.2283	0.11	Negligible

question asked how long they had programmed using the object oriented paradigm, considering course work in this case. We present distribution of the answers in Fig. 6. The participants of FinG 2 had two years or more of experience than the participants of FinG 1.

Due to this difference, we grouped participants of FinG 1 and FinG 2 in different classes. We adopted the classification presented by Höst et al. (2005). They addressed experience as context variable in empirical studies and proposed a classification, which we show in Table 14. The scale in the table is ordinal, a higher value corresponds to more experience than a lower value (E1 < E2, E2 < E3, and so on). It is important to note the authors did not present the classes as a rigid classification, but as a starting point to analysis of experience. As all participants of FinG 1 were undergraduate students, they are all in the E1 class. Participants of FinG 2 were classified in-between the E2 and E5 classes. In spite being graduate students, their profiles indicated most of them were or had been professionals.

5.2.1 Agreement among the participants

Table 15 shows agreement coefficient values for FinG 1 and FinG 2 for each program. In most of the programs, the agreement is the same for FinG 1 and FinG 2, except for Chess program. In this case, the agreement is higher in FinG 2 (INCR). For the Jackut program in FinG 1, and Tic Tac Toe in FinG 2, the number of participants is different of the other cases. This happened because the participants marked in the form a class as a candidate, but they did not fill the name of the classes. We removed these data to avoid affecting the test.

5.2.2 Agreement between the participants and the oracle

To perform statistical tests comparing accuracy values of FinG 1 and FinG 2, we defined the null hypothesis. We expected more experienced participants succeed more than less experienced participants. Then, our null hypothesis is:

- *H0: there is no difference of accuracy values between participants of FinG 1 and FinG 2*

Table 16 shows statistical tests. We highlighted Mann-Whitney p-value because we rejected the null hypothesis. We also highlighted the magnitude of the Cliff's Delta value because it showed some magnitude (despite small) on the effect size.

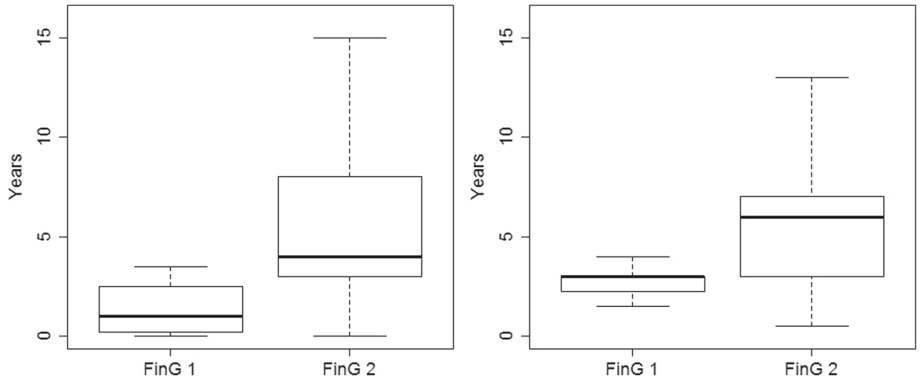

Fig. 6 Years of experience, for FinG 1 and FinG 2's participants

Table 14 Class of experiences of participants (Höst et al. 2005)

Category	Description
E1	Undergraduate student with less than 3 months recent industrial experience
E2	Graduate student with less than 3 months recent industrial experience
E3	Academic with less than 3 months recent industrial experience
E4	Any person with recent industrial experience, between 3 months and 2 years
E5	Any person with industrial experience for more than 2 years

5.2.3 Analysis

From the results, we consider that *the experience impacts agreement on god class detection, but not definitively*. The hypothesis test and the magnitude of the effect size[5] confirmed that there was a difference on the accuracy values with respect to the oracle for both more and less experienced participants groups. However, most cases of agreement among the participants were classified as the "same" for both groups. Only in one case the agreement level was higher for more experienced developers. These findings may suggest that there are other human aspects affecting god class detection.

5.3 Developer knowledge

In this section, we present the results related to developer knowledge variable. In this case, we based our analysis on a questionnaire applied during the training of FinG 2 and FinG 3. We discussed the questionnaire in Section 3.8. To evaluate the knowledge, we analyzed answers of the questionnaire. We defined a template, before reading the answers. Then, we classified the answers as follows:

1. Wrong answer (distant of the central idea)
2. Some correct propositions, but without capturing the main idea

Table 15 Finn agreement test for FinG 1 and FinG 2 experiments

Program	Experiment	#class	#part	Finn	*p*-value	Agreement level
Monopoly	FinG 1	4	11	0.636	7.4e-05	Substantial
	FinG 2	8	25	0.732	2.51e-26	Substantial
	Evolution of agreement level from FinG 1 to FinG 2					SAME
Tetris	FinG 1	8	11	0.682	8.12e-10	Substantial
	FinG 2	10	25	0.78	2.56e-40	Substantial
	Evolution of agreement level from FinG 1 to FinG 2					SAME
Tic Tac Toe	FinG 1	4	11	0.691	7.94e-06	Substantial
	FinG 2	5	24	0.713	3.12e-15	Substantial
	Evolution of agreement level from FinG 1 to FinG 2					SAME
Chess	FinG 1	10	11	0.716	1.22e-13	Substantial
	FinG 2	10	25	0.803	8.14e-45	Almost perfect
	Evolution of agreement level from FinG 1 to FinG 2					INCR
Jackut	FinG 1	8	10	0.722	1.67e-10	Substantial
	FinG 2	6	25	0.66	5.65e-15	Substantial
	Evolution of agreement level from FinG 1 to FinG 2					SAME
Solitaire	FinG 1	10	11	0.702	7.3e-13	Substantial
	FinG 2	6	25	0.649	2.61e-14	Substantial
	Evolution of agreement level from FinG 1 to FinG 2					SAME

Table 16 Hypothesis test for analysis of developer experience, considering FinG 1 and FinG 2

Shapiro-Wilk				Mann-Whitney		Cliff's Delta effect size	
FinG 1		FinG 2					
W	*p*-value	W	*p*-value	W	*p*-value	Delta	Magnitude
0.8419	8.066e-07	0.7807	1.163e-13	3974.5	0.02735	0.18	small

3. Capturing main idea, but without a good explanation
4. Capturing main idea, with a good explanation (perfect answer)

From our template, we scored the answers of each participant from 3 (three wrong answers) to 12 (three perfect answers). We grouped the participants according to three different thresholds scores: i) 70%, ii) 80% and, iii) the median. All results impacted our analysis in the same way. In this paper, we show results considering the threshold score as 70%. We consider 70% as a good score to indicate the participants have an appropriate knowledge on the code smell subject.

We considered participants with scores equal or above 70% as with "good" knowledge. In opposition, we considered participants with scores under 70% as "bad" knowledge. In FinG 2, there were 15 participants with "good" knowledge, and nine participants with "bad" knowledge. Note that, the total of participants considered was 24: one of them did not send us his/her questionnaire. Due to this, we removed the participant from this analysis. In FinG 3, there were nine participants with "good" knowledge, and 16 participants with "bad" knowledge.

5.3.1 Agreement among the participants

Table 17 shows agreement coefficient values for FinG 2 according to the knowledge of the participants. There were two cases where the agreement level decreased, when participants had "good" knowledge: Tic Tac Toe and Chess programs. For the other four cases, the agreement level was the same.

Table 18 has the same structure of the previous table, but it shows the agreement values among participants of FinG 3. The main difference on the setup of Fing 2 and FinG 3 is the software size. FinG 3 used four medium size programs. In two cases, the agreement level among the participants with "good" knowledge increased. For the other two cases, the agreement level was the same.

5.3.2 Agreement between the participants and the oracle

We expected that participants with "good" knowledge succeeded more than participants with "bad" knowledge. Then, we defined the null hypothesis as:

- *H0: there is no difference of accuracy values between participants with "good" knowledge and "bad" knowledge*

In Table 19, we show statistical tests for both FinG 2 and FinG 3 experiments. In both cases, we could not reject the null hypothesis. The magnitude on the effect size, for both experiments, was negligible.

5.3.3 Analysis

We highlight conflicting results in FinG 2. There were two cases where the agreement level was lower for participants with "good" knowledge in FinG 2. However, we had oppo-

Table 17 Finn agreement test for FinG 2, grouped by participants' knowledge

Program	Knowledge	#class	#part	Finn	*p*-value	Agreement level
Monopoly	Bad	8	9	0.681	4.02e-08	Substantial
	Good	8	15	0.757	6.88e-18	Substantial
	From bad knowledge to good knowledge evolution					SAME
Tetris	Bad	10	9	0.756	3.71e-13	Substantial
	Good	10	15	0.775	9.43e-24	Substantial
	From bad knowledge to good knowledge evolution					SAME
Tic Tac Toe	Bad	5	9	1	0	Almost perfect
	Good	5	15	0.611	9.02e-07	Substantial
	From bad knowledge to good knowledge evolution					DECR
Chess	Bad	10	9	0.856	1.3e-20	Almost perfect
	Good	10	15	0.779	3.7e-24	Substantial
	From bad knowledge to good knowledge evolution					DECR
Jackut	Bad	6	9	0.63	2.02e-05	Substantial
	Good	6	15	0.683	2.94e-10	Substantial
	From bad knowledge to good knowledge evolution					SAME
Solitaire	Bad	6	9	0.63	2.02e-05	Substantial
	Good	6	15	0.657	2.66e-09	Substantial
	From bad knowledge to good knowledge evolution					SAME

site findings in FinG 3, where there were two cases in which the agreement was higher for participants with "good" knowledge. Besides, the hypothesis test and effect size were not conclusive.

The dependent variable, in this case, might be affected by the different types of software used in FinG 2 and FinG 3. Remember that the software used in FinG 3 is more complex than the software used in FinG 2. Clearly, software size is an important confounding factor, but the findings also indicate that the developer knowledge is more relevant on evaluation of more complex software. Due to this, although we do not consider the evidences are very strong, we believe knowledge also impacts agreement to a certain level, and the subject should be further analyzed.

Table 18 Finn agreement test for FinG 3, grouped by participants' knowledge

Program	Knowledge	#class	#part	Finn	*p*-value	Agreement level
JMoney	Bad	12	16	0.506	1.39e-09	Moderate
	Good	12	9	0.556	4.78e-07	Moderate
	From bad knowledge to good knowledge evolution					SAME
JParse	Bad	12	16	0.381	1.51e-05	Fair
	Good	12	9	0.481	2.54e-05	Moderate
	From bad knowledge to good knowledge evolution					INCR
Quilt	Bad	12	16	0.593	7.65e-14	Moderate
	Good	12	9	0.611	1.04e-08	substantial
	From bad knowledge to good knowledge evolution					INCR
Squirrel	Bad	12	16	0.501	2.04e-09	Moderate
	Good	12	9	0.574	1.46e-07	Moderate
	From bad knowledge to good knowledge evolution					SAME

Table 19 Hypothesis test for analysis of developer knowledge, considering FinG 2 and FinG 3

FinG	Shapiro-Wilk				Mann-Whitney		Cliff's Delta effect size	
	Bad knowledge		Good knowledge					
	W	*p*-value	W	*p*-value	W	*p*-value	Delta	Magnitude
2	0.7147	7.807e-09	0.7995	1.002e-09	2673.5	0.1921	0.12	Negligible
3	0.892	4.049e-05	0.896	0.002658	1080.5	0.6025	0.06	Negligible

5.4 Software size

In this section, we present results related to our analysis of how software size impacts agreement on god class detection. We compared agreement of FinG 2 against FinG 3. We did this because the main difference on experimental setup was the type of software adopted. In FinG 2, we used simple software; in FinG 3, we used medium sized – more complex – software.

5.4.1 Agreement among the participants

Table 20 shows results of agreement among the participants of FinG 2. In the same way, we show results for FinG 3, in Table 21. Once software adopted in the experiments were different, we did not analyze the evolution. Instead, we compared the general tendency of agreement level in both FinG 2 and FinG 3 experiments. In FinG 2, there was one almost perfect level of agreement for Chess software. For all other values, agreement level was substantial. In FinG 3, there was one substantial agreement level, for Quilt software. For all other values, agreement level was moderate.

5.4.2 Agreement between the participants and the oracle

We consider that participants detecting god class on simpler software succeed more than participants detecting god class on more complex software. The null hypothesis is:

- *H0: There is no difference of accuracy values between participants of FinG 2 and FinG 3*

Table 22 shows statistical test. In this case, we rejected the null hypothesis. We highlighted the Cliff's Delta value because it showed some magnitude (medium) on the effect size.

5.4.3 Analysis

All analysis procedure we adopted indicates that *software size impacts agreement on god class detection*. In our experiments, participants agreed and succeed more detecting god class in simpler software (FinG 2) than in more complex software (FinG 3). We found strong evidences of this. First, most of the level of agreement was substantial in FinG 2

Table 20 Finn agreement test and level for FinG 2

Program	#class	#part	Finn	*p*-value	Agreement level
Monopoly	8	25	0.732	2.51e-26	Substantial
Tetris	10	25	0.78	2.56e-40	Substantial
Tic Tac Toe	5	24	0.713	3.12e-15	Substantial
Chess	10	25	0.803	8.14e-45	Almost perfect
Jackut	6	25	0.66	5.65e-15	Substantial
Solitaire	6	25	0.649	2.61e-14	Substantial

Table 21 Finn agreement test and level for FinG 3

Program	#class	#part	Finn	*p*-value	Agreement level
JMoney	12	25	0.511	1.04e-14	Moderate
JParse	12	25	0.43	4.14e-10	Moderate
Quilt	12	25	0.608	1.63e-22	Substantial
Squirrel	12	25	0.529	6.29e-16	Moderate

(one almost perfect) and just moderate in FinG 3 (one substantial). Second, it was possible to reject the null hypothesis. Third, the magnitude of the effect size (which was classified as medium) reinforced our idea that participants succeed more detecting god classes in simpler software. This set of evidences reinforces our previous conjecture about software size as a confounding factor in the analysis of the developer knowledge.

5.5 Developer training

The main difference between the experimental setup of FinG 3 and FinG 4 was the training. After the analysis of the other variables, we concluded that the conceptualization of god class is more dependent of human traits than of technical aspects. Due to this, we conjectured if it would be possible to affect the agreement aligning the personal thresholds on god class detection. In order to test this idea, the training of FinG 4 went beyond reading, including training with golden examples and live discussions.

5.5.1 Agreement among the participants

Table 23 shows the agreement coefficient among participants of FinG 3 and FinG 4. In three cases, the agreement level in FinG 4 was higher than in FinG 3.

5.5.2 Agreement between the participants and the oracle

According previous discussion, we expect participants who are trained with "golden" examples, discussing personal thresholds, succeed more than participants trained only reading books. Due to this, the null hypothesis is:

- *H0: There is no difference of accuracy values between participants of FinG 3 and FinG 4*

Table 24 shows the statistical test. As in our previous analysis, we rejected the null hypothesis and the Cliff's Delta value showed some magnitude (despite small) on the effect size.

5.5.3 Analysis

We found that *participants discussing the conceptualization and personal thresholds based on examples agree and succeed more than participants in a reading-based training.* First, the level of agreement increased for most cases in FinG 4. Only in one case the level

Table 22 Hypothesis tests for software size, considering FinG 2 and FinG 3 accuracy values

Shapiro-Wilk				Mann-Whitney		Cliff's Delta effect size	
FinG 2		FinG 3					
W	*p*-value	W	*p*-value	W	*p*-value	Delta	Magnitude
0.7807	1.163e-13	0.8934	6.978e-07	10183	4.424e-07	0.37	Medium

Table 23 Finn agreement test for FinG 3 and FinG 4

Program	Experiment	#class	#part	Finn	*p*-value	Agreement level
JMoney	FinG 3	12	25	0.511	1.04e-14	Moderate
	FinG 4	12	17	0.64	8.12e-18	Substantial
	From FinG 3 to FinG 4 evolution					INCR
JParse	FinG 3	12	25	0.43	4.14e-10	Moderate
	FinG 4	12	17	0.515	1.61e-10	Moderate
	From FinG 3 to FinG 4 evolution					SAME
Quilt	FinG 3	12	25	0.608	1.63e-22	Moderate
	FinG 4	12	17	0.841	1.41e-43	Almost perfect
	From FinG 3 to FinG 4 evolution					INCR
Squirrel	FinG 3	12	25	0.529	6.29e-16	Moderate
	FinG 4	12	17	0.718	7.86e-25	Substantial
	From FinG 3 to FinG 4 evolution					INCR

of agreement was the same in FinG 3 and FinG 4. And even in this case, the Finn coefficient was higher in FinG 4 than in FinG 3 (see values for the JParse software in Table 23). Second, it was possible to reject the null hypothesis. Third, the magnitude of the effect size reinforced our idea that the training impacts on god class detection. We conjecture that this finding reinforces our beliefs that conceptualization of code smells is the main cause of inconsistencies of empirical studies on the topic. We believe that the training based on examples and discussions is the best approach to control human traits, which significantly affects the agreement on god class detection.

6 Discussion

In this section, we present our main findings considering all previous analyses of the current paper and of other papers based on FinG family. First, we summarize all main findings considering each factor analyzed in the current paper. Second, we present previous findings from other papers based on FinG family. Finally, we present an overall discussion considering all results.

6.1 Findings from the analyses presented in the current paper

Developer's experience, training and even knowledge impacted the agreement, but the use of software visualization did not. These evidences show that god class detection is more related to human traits than to an overall understanding of the code design. We believe this finding is transversal to other types of code smell, because the human aspects are related to subjective evaluation of the design quality, which occurs with other types of smell, in different level of intensity. Based on this, we consider studies focused on smell detection that disregard human aspects might be misdirected. Sjøberg et al. (2013) also agree with this perception.

Table 24 Hypothesis tests for analysis of developer training, considering FinG 3 and FinG 4

Shapiro-Wilk				Mann-Whitney		Cliff's Delta effect size	
FinG 3		FinG 4					
W	*p*-value	W	*p*-value	W	*p*-value	Delta	Magnitude
0.8934	6.978e-07	0.8367	3.531e-07	2392	0.0008652	0.3	Small

Software size also did affect the agreement. Agreement was lower in medium size than in small size software. This is evidence that context factors, other than human traits, affect the conceptualization of code smells. This dimension must also be further explored in future experiments.

6.2 Findings from other papers based on FinG family

We presented a preliminary analysis of FinG 1 based on different type of data (Santos et al. 2013) and (Santos et al. 2014). Despite we asked participants to indicate their candidate god classes, we also asked them to indicate why they think a class is a god class. Moreover, we logged the actions they realized on the Eclipse IDE. We then investigated: i) the agreement (based on the god classes indicated by the participants and the oracle); ii) the strategies adopted by participants identifying a god class (based on the log of actions realized on the Eclipse IDE); and iii) decision drivers indicating why participants consider a class as a god class. The other papers we had presented deep the analysis of the strategies adopted by the participants and the decision drivers: Santos and Mendonça (2014) considered the results of FinG 1 and FinG 2; and (Santos and Mendonça 2015) considered the results of FinG 1, FinG 2 and FinG 3.

We highlight the main findings related to the agreement, strategies and decision drivers as follows. From the analysis of the agreement in FinG 1, we noted low level of agreement. We also noted that the use of the visualization tool did not impact FinG 1 results. This finding is reinforced by the analysis that we present in the current paper, Section 5.1. These findings motivated us to deep the analysis performing the other controlled experiments of FinG family, up to now. From the analysis of the strategies adopted by FinG 1/FinG 2's participants, we found that the observation of coupling is more relevant than the observation of attributes like LOC or complexity and the hierarchical relation among classes on god class detection. We also noted that reading source code is important, even with visual resources enhancing the general comprehension of the software. From the analysis of the decision drivers, we found that "class is high complex" and "method is misplaced" are the stronger drivers. We also found the agreement on drivers' choice is low. Another finding is: some important drivers are dependent of alternative support. In our case, "dependency" was an important driver only when visual resources were permitted.

6.3 Overall discussion

Overall, the results presented in the previous and current paper about FinG family evidence the relevance of subjectivity on god class detection. We believe this finding is transversal to other types of code smells. The absence of an extensive set of empirical evidence on the topic might misdirect some researches that consider software quality is mainly related to measurable attributes.

For example, the use of heuristics for automatic detection of smells, which are created by humans, should be adjusted for specific contexts. The proposition of new heuristics should be presented after the evaluation of the relationship between the current heuristics and the context in which the smells are used (human traits there included). This does not seem to be the current trend on the subject. In their systematic mapping study, Zhang et al. (2011) noted that the most of studies in code smells focus on tools and methods for automatic detection. Evaluating some of these and other more recent studies, we

noted that they propose heuristics for automatic detection of code smells disregarding the context where apply them.

To get a better comprehension of the smell effect, it is necessary to get a better comprehension of how human evaluate design quality. We call this issue the *"code smell conceptualization problem"*. The *code smell conceptualization problem* involves the discussion about ensuring same comprehension of the smell concepts and similar personal criteria or thresholds. We consider that two main trails should be followed to solve this problem: i) to deepen the evaluation of the human aspects and how they affect smell detection; and ii) to evaluate cognitive aspects on smell detection, which is related to program comprehension, and requires knowledge both in computer science and cognitive psychology (Jonathan and Maletic 2008).

Table 25 summarizes the main findings presented in this paper. At the top of the table, we present the main findings related to the addressed factors by FinG. At the bottom, we summarize the overall discussion that we propose.

7 Threats to validity

Our analysis of threats to validity of the study was based on the classification adopted by Wohlin et al. (2012).

External validity. Our first threat fits in the "interaction of selection and treatment" subcategory and is related to the fact that the experiments were carried out in an in-vitro setting. Participants of FinG 2, FinG 3 and FinG 4 were graduate students; and participants of FinG 1 were undergraduates. One aspect mitigates the threat: most participants of FinG 2, FinG 3 and FinG 4 had some professional software development experience. Moreover, our focus was the investigation of the impact of human traits on smell detection, as opposed to the quality of the evaluation of the smell detection, which would arguably be more affected by the profile of the participants.

Other threat to external validity fits in the "interaction of setting and treatment" subcategory. In this case, the threat is the type of software used in FinG 1 and FinG 2. We

Table 25 Summary of the main findings

Factor	Findings
Design comprehension	A design comprehension tool support does not improve agreement on god class detection
Developer experience	The experience impacts agreement on god class detection, but not definitively
Developer knowledge	It should be further analyzed
Software size	Software size impacts agreement on god class detection
Developer training	Participants discussing the conceptualization and personal thresholds based on examples agree and succeed more than participants in a reading-based training
Overall discussion	God class detection is more related to human traits than to an overall understanding of the code design
	Other context factors not related to the human traits, such as software size, also affect human conceptualization of smells. This issue should be further investigated
	New heuristics for smell detection should be presented after the evaluation of the relationship between two aspects: 1) the current heuristics and 2) the context in which the smells are used (human traits there included)
	The area should addresses the *code smell conceptualization problem*, which involves the discussion about ensuring same comprehension of the smell concepts and similar personal criteria or thresholds

adopted six small programs. We mitigated this threat by using medium programs in FinG 3 and FinG 4. In the same subcategory, we also highlight that the participants identified bad smells in an unknown family of software, which is not the common. We emphasize that this is an important threat impacting our conclusion because it is much harder to find bad smells in an unknown software. To mitigate the threat, we argue that: i) the domain of all software objects adopted in our study are well known; and ii) all participants were impacted in the same way by it (none of participants previously knows the software objects adopted in the experiments).

Internal validity. There is a threat of "ambiguity about direction of causal influence". In FinG 1, the training on god class was influenced by the view of the experimenter (also the trainer). To minimize this effect, we limited the time of training in the god class concept and adopted the support questions used in the Schumacher's experiment (Schumacher et al. 2010) to guide participants on the god class detection during the experiment. We also mitigated this threat in FinG 2, FinG 3 and FinG 4. Instead of a presentation by the experimenter, participants read reference books to acquire or improve their knowledge on the topic. Another threat in same subcategory was the training in the visualization tool in FinG 1 and FinG 2. In our feedback form, participants indicated that the quality of training was good, in general. However, we cannot confirm that it was sufficient to prepare participants in the use of visualization. We consider this a weak empirical evidence of the validity of the training. However, we are confident about the answers because the software visualization tool is very simple and some participants gave positive comments after the experiments. Finally, the last threat in the same subcategory is the use of different researchers as the oracle of FinG 1-2 and FinG 3-4. One can argues that this might to affect the god class detection. We consider this threat is not significantly impacting our analyses because: i) we consider subjectivity is expected on code smell detection; ii) all oracle researchers have experience on the topic; iii) the main experimenter is one of the oracle researchers in both FinG 1-2 and FinG 3-4 experiments; and iv) the definition of the oracle was mainly based on discussion between the researchers.

Another subcategory of the internal validity is "maturation". Participants could be affected because they did the same task over six programs, in FinG 1 and FinG 2, and four "more complex" programs, in FinG 3 and FinG 4. They may have learned and worked faster as they progressed on the experiments. On the other hand, they could be negatively affected by boredom. We consider maturation a weak threat because the experiments were performed in 2 hours, on average. We consider this a reasonable period of time to do a task in a balanced way.

Conclusion validity. In the "reliability of treatment implementation" threat, we have to consider that a participant who could have used visualization may have completely disregarded the resources, which would impact the analysis of FinG 1 and FinG 2. However, we logged and checked actions performed in the Eclipse IDE. We observed that only one participant in FinG 1, and two participants in FinG 2 did not use the visualization tool resources when they were available. This was done by the participants own choice and did not affect the results significantly. This issue is discussed in detail in Santos et al. (2014). We also have to consider if the visualization tool was appropriated for the identification of useful attributes for god class detection. We are confident about this, because of former experiments with the tool (Carneiro et al. 2010; Carneiro and Mendonça 2013, 2014) and the discussions that occurred during the training of FinG 1 and FinG 2. Another threat in

the same subcategory was related to the evaluation of the questionnaire answers, which was part of the training in FinG 2 and FinG 3. The evaluation of the answers of the participants was subjective. We mitigated this threat comparing the answers with a template defined previously. Moreover, the questions were very simple.

Another subcategory of the conclusion validity is "random heterogeneity of subjects". We analyzed how software size impacts agreement on god class detection comparing results of FinG 2 and FinG 3. We have to consider that, despite differences on software size, the population of the experiments was formed by different participants subjects. We consider this a weak threat because the profile of the participants is similar. For both FinG 2 and FinG 3 experiments, most of the participants were graduate students with some years working with software development in software industry or research projects. There were few cases (six considering both experiments) where the participants had no professional experience on software development. Even in that case, the participants declared to have knowledge about OO programming.

Construct validity. In the "inadequate preoperational explication of constructs", we have to consider the god class concept we adopted involves conceptual definitions of god class, brain class and large class. In order to mitigate this threat, we considered the adoption of the support questions and the effort on the training for all experiments. This was added to the presentation, reading of books, and questionnaire answers. Moreover, on FinG 4 training, it was possible to discuss with participants their conceptualization about god class.

Another threat that we found in the same subcategory is related to the material support considered in the analysis of some factors. For example, the analysis of the level of knowledge was based on the answers from a questionnaire defined by us. Due to the simplicity of the questions, we defined the questionnaire based on our own perceptions about what is important in order to capture the developer knowledge on the subject. In the same way, we defined the questions of the characterization form used in the analysis of the developer experience; and in the definition of the SourceMiner exercise, which trained the participants on the visualization tool. We mitigated this threat by adapting some of the material from works previously published in the literature (Schumacher et al. 2010; Novais et al. 2012) and discussing about them with other researchers from our research community.

8 Conclusion

This paper aims to improve the understanding of factors affecting human evaluation of code smells. To do this, we carried out a family of four controlled experiments. The experiments were defined in an iterative process. Insights of one experiment were used in definition of the next ones, in order to batter control factors and explore insights gained in the previous experimental cycle (Mendonça et al. 2008). Specifically, we addressed cause-effect relation between five factors and the agreement on god class detection. The factors were: tool support for design comprehension, developer experience, developer knowledge, size software and training.

Due to the difficulties of controlling factors affecting smell detection, the software engineering community has more and more focused of the use of software metrics for smell detection. The discussion we provided here, which we called the *code smell conceptualization problem*, shows that the problem is complex and the community must ensure similar personal criteria and thresholds on smell detection.

Tool support for design comprehension (which was obtained by the use of software visualization) was the only factor did not impact agreement on god class detection. Developer experience and training strongly impacted the agreement. This evidences god class detection is more related to human traits than to an overall understanding of the code design. We believe this finding is transversal to other code smells.

As smell detection is strongly affected by human aspects, and other context variables, the heuristics based on metrics, that are being used by the software engineering community currently, should be always checked and adjusted for specific situations. This also raises out the need for more studies focusing on the identification and characterization of factors that affects the perception and impact of code smells, specially human aspects.

We showed that training impacts god class detection. As this is a controllable factor, we propose training based on "golden" examples and discussions between developers to align their conceptualization about code smells in software organizations. This way they may seize some control over the smell conceptualization problem.

We also believe that the discussion on the effectiveness of adoption of code smell concepts in software development needs to evolve. We have already published other partial results of our work (Santos et al. 2013, 2014; Santos and Mendonça 2014, 2015), exploring detection effort, code smell decision drivers and detection strategies. To support replications of our experiments, we make our experimental package available to the software engineering research community[6].

To address the limitations of this study and to further develop it, we are planning other experiments addressing other code smells. We are also performing a systematic review on the subject to be published in another paper. This systematic review will address an important aspect related to the set of empirical results now available in the area. The aspect is related to difficulties in generalizing the findings because of: i) the variety of factors being investigated and; ii) differences on the context in which the experiments have been presented.

Endnotes

[1] https://marketplace.eclipse.org/content/jdeodorant

[2] https://marketplace.eclipse.org/content/incode-helium

[3] Eclipse IDE -http://www.eclipse.org/ and SourceMiner - http://www.sourceminer.org/

[4] We consider as a golden example a class that all developers consider as a god class, because it has many lines of code or many methods addressing different issues on the software.

[5] As discussed in Section 4.3, we are considering small magnitude on the effect size as evidence reinforcing our hypothesis because of the small number of data points in our samples.

[6] http://wiki.dcc.ufba.br/LES/JoseAmancio

Abbreviations

FinG: Family of experiments *Finding God Class*; LOC: Lines of code; UFBA: Federal University of Bahia

Acknowledgements

This work was partially supported by SECTI-BA (Bureau of Science and Technology of Bahia) - Fraunhofer Project Center for Software & Systems Eng. agreement 2012/001.

Funding

This work was partially supported by the State of Bahia through SECTI-UFBA-Fraunhofer's Cooperation Agreement 2012-1.

Availability of data and materials
The experimental package for all experiments are available at http://wiki.dcc.ufba.br/LES/JoseAmancio.

Authors' contributions
JAMS was the main experimenter. He planned and carried out the experiments and analyses presented in the paper. JBR-J and MGdeM served as consultants, evaluating the setup, results and analyses for all experiments presented in the paper. They also acted as reviewers, supporting the paper's writing. All authors read and approved the final manuscript.

Ethics approval and consent to participate
All participants of the experiments signed a form consenting the research publishing. Moreover, in order to attend ethical issues on Empirical Software Engineering, we followed principles proposed by (Vinson and Singer 2008).

Consent for publication
All authors consent to the publication of the manuscript in JSERD.

Competing interests
The authors declare that they have no competing interests.

Publisher's Note

Author details
[1]Technology Department, State University of Feira de Santana, Avenida Transnordestina, s/n - Novo Horizonte, CEP 44036-900, Feira de Santan, Bahia, Brazil. [2]Exact Science Department, State University of Feira de Santana, Feira de Santana, Bahia, Brazil. [3]Fraunhofer Project Center for Software & Systems Engineering, Federal University of Bahia, Salvador, Bahia, Brazil. [4]Department of Computer Science, Federal University of Bahia, Salvador, Bahia, Brazil.

Received: 16 June 2017 Accepted: 31 October 2017
Published online: 28 November 2017

References
Abbes M, Khomh F, Guéhéneuc YG, Antoniol G (2011) An empirical study of the impact of two antipatterns, blob and spaghetti code, on program comprehension. In: Proc. of 15th European Conference on Software Maintenance and Reengineering (CSMR). pp 181–190
Ahmed I, Mannan UA, Gopinath R, Jensen C (2015) An empirical study of design degradation: How software projects get worse over time. In: 2015 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM). IEEE Computer Society, Washington, DC. pp 1–10
Anda B (2007) Assessing software system maintainability using structural measures and expert assessments. In: Proc. of the 23rd IEEE International Conference on Software Maintenance (ICSM). IEEE Computer Society, Washington, DC. pp 204–213
Basili V, Shull F, Lanubile F (1999) Building knowledge through families of experiments. IEEE Trans Serv Comput 25(4):456–473
Carneiro GF, Mendonça MG (2013) Sourceminer - a multi-perspective software visualization environment. In: Proc. of the 15th International Conference on Enterprise Information Systems (ICEIS). Springer, Berlin
Carneiro G, Mendonça M (2014) Sourceminer: Towards an extensible multi-perspective software visualization environment. In: Hammoudi S, Cordeiro J, Maciaszek LA, Filipe J (eds). Enterprise Information Systems, Lecture Notes in Business Information Processing, vol 190. Springer International Publishing, Cham. pp 242–263
Carneiro G, Silva M, Maia L, Figueiredo E, Sant'Anna C, Garcia A, Mendonça M (2010) Identifying code smells with multiple concern views. In: Proc. of the 1th Brazilian Conference on Software: Theory and Practice (CBSOFT). IEEE Computer Society, Washington, DC. pp 128–137
Carver J, Jaccheri L, Morasca S, Shull F (2003) Issues in using students in empirical studies in software engineering education. In: Proc. of the 9th International Software Metrics Symposium. IEEE Computer Society, Washington, DC. pp 239–249
Cliff N (1996) Ordinal Methods for Behavioral Data Analysis. Erlbaum, Mahwah. ISBN:9780805813333
Cohen J (1960) A coefficient of agreement for nominal scales. Educ Psychol Meas 20:37–46
de Alwis B, Murphy GC (2006) Using visual momentum to explain disorientation in the eclipse ide. In: Proc. of the Visual Languages and Human-Centric Computing (VLHCC). IEEE Computer Society, Washington, DC. pp 51–54
Dybå T, Sjøberg DI, Cruzes DS (2012) What works for whom, where, when, and why?: On the role of context in empirical software engineering. In: Proc. of the 6th International Symposium on Empirical Software Engineering and Measurement (ESEM). ACM, New York. pp 19–28
Feinstein AR, Cicchetti DV (1990) High agreement but low kappa: I. the problems of two paradoxes. J Clin Epidemiol 43(6):543–549
Fernandes E, Oliveira J, Vale G, Paiva T, Figueiredo E (2016) A review-based comparative study of bad smell detection tools. In: Proceedings of the 20th International Conference on Evaluation and Assessment in Software Engineering, ACM, New York, NY, USA, EASE '16. ACM, New York. pp 18:1–18:12
Finn RH (1970) A note on estimating the reliability of categorical data. Educ Psychol Meas 30:71–76
Fleiss J, et al (1971) Measuring nominal scale agreement among many raters. Psychol Bull 76(5):378–382
Fontana FA, Braione P, Zanoni M (2012) Automatic detection of bad smells in code: An experimental assessment. J Object Technol 11(2):5:1–38
Fowler M (1999) Refactoring: improving the design of existing code. Addison-Wesley Longman Publishing Co., Inc., Boston

Fu S, Shen B (2015) Code bad smell detection through evolutionary data mining. In: 2015 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM). IEEE Computer Society, Washington, DC. pp 1–9
Gwet K (2002) Kappa statistic is not satisfactory for assessing the extent of agreement between raters. Stat Methods Inter-rater Reliab Assess 1(1):1–5
Höst M, Regnell B, Wohlin C (2000) Using students as subjects: A comparative study of students and professionals in lead-time impact assessment. Empir Softw Eng 5(3):201–214
Höst M, Wohlin C, Thelin T (2005) Experimental context classification: incentives and experience of subjects. In: Proc. of the 27th International Conference on Software Engineering (ICSE). IEEE Computer Society, Washington, DC. pp 470–478
Jedlitschka A, Ciolkowski M, Pfahl D (2008) Reporting experiments in software engineering. In: Shull F, Singer J, Sjberg DIK (eds). Guide to Advanced Empirical Software Engineering. Springer, London. pp 201–228
Johnson B, Shneiderman B (1991) Tree-maps: a space-filling approach to the visualization of hierarchical information structures. In: Proc. of IEEE Conference on Visualization. IEEE Computer Society, Washington, DC. pp 284–291
Jonathan I, Maletic HK (2008) Expressiveness and effectiveness of program comprehension: Thoughts on future research directions. In: Frontiers of Software Maintenance (FoSM). IEEE Computer Society, Washington, DC. pp 31–37
Juristo N, Vegas S (2009) Using differences among replications of software engineering experiments to gain knowledge. In: Proc. of the 3rd International Symposium on Empirical Software Engineering and Measurement (ESEM). IEEE Computer Society, Washington, DC. pp 356–366
Kersten M, Murphy GC (2006) Using task context to improve programmer productivity. In: Proc. of the 14th ACM SIGSOFT International Symposium on Foundations of Software Engineering (SIGSOFT '06/FSE-14). ACM, New York. pp 1–11
Khomh F, Vaucher S, Gueheneuc YG, Sahraoui H (2009) A bayesian approach for the detection of code and design smells. In: Proc. of the 9th International Conference on Quality Software (QSIC). IEEE Computer Society, Washington, DC. pp 305–314
Kitchenham B, Madeyski L, Budgen D, Keung J, Brereton P, Charters S, Gibbs S, Pohthong A (2017) Robust statistical methods for empirical software engineering. Empir Softw Engg 22(2):579–630
Kreimer J (2005) Adaptive detection of design flaws. Electron Notes Theor Comput Sci 141(4):117–136
Landis JR, Koch GG (1977) The measurement of observer agreement for categorical data. Biometrics 33(1):159–174
Lanza M, Ducasse S (2003) Polymetric views - a lightweight visual approach to reverse engineering. IEEE Trans Softw Eng 29(9):782–795
Lanza M, Marinescu R (2005) Object-Oriented Metrics in Practice. Springer-Verlag New York, Inc., Secaucus
Li W, Shatnawi R (2007) An empirical study of the bad smells and class error probability in the post-release object-oriented system evolution. J Syst Softw 80(7):1120–1128
Linares-Vásquez M, Klock S, McMillan C, Sabané A, Poshyvanyk D, Guéhéneuc YG (2014) Domain matters: Bringing further evidence of the relationships among anti-patterns, application domains, and quality-related metrics in java mobile apps. In: Proceedings of the 22Nd International Conference on Program Comprehension, ICPC 2014. ACM, New York. pp 232–243
Mäntylä M (2005) An experiment on subjective evolvability evaluation of object-oriented software: explaining factors and interrater agreement. In: Proc. of the 4th International Symposium on Empirical Software Engineering (ISESE). IEEE Computer Society, Washington, DC
Mäntylä M, Lassenius C (2006a) Subjective evaluation of software evolvability using code smells: An empirical study. Empir Softw Eng 11(3):395–431
Mäntylä MV, Lassenius C (2006b) Drivers for software refactoring decisions. In: Proc. of 5th International Symposium on Empirical Software Engineering (ISESE). Kluwer Academic Publishers, Hingham. pp 297–306
Mäntylä MV, Vanhanen J, Lassenius C (2004) Bad smells humans as code critics. In: Proc. of the 20th IEEE International Conference on Software Maintenance (ICSM). IEEE Computer Society, Washington, DC. pp 399–408
Mendonça M, Maldonado J, de Oliveira M, Carver J, Fabbri S, Shull F, Travassos GH, Hohn E, Basili V (2008) A framework for software engineering experimental replications. In: Proc. of the 13th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS). IEEE Computer Society, Washington, DC. pp 203–212
Meyer B (1988) Object-Oriented Software Construction. 1st edn. Prentice-Hall, Inc., Upper Saddle River
Moha N, Guéhéneuc Y, Duchien L, Le Meur A (2010) Decor: A method for the specification and detection of code and design smells. Softw Eng. IEEE Trans 36(1):20–36
Moonen L, Yamashita A (2012) Do code smells reflect important maintainability aspects? In: Proc. of 28th the International Conference on Software Maintenance (ICSM). IEEE Computer Society, Washington, DC. pp 306–315
Murphy-Hill E, Black AP (2010) An interactive ambient visualization for code smells. In: Proc. of the 5th International Symposium on Software visualization (SOFTVIS). ACM, New York. pp 5–14
Novais R, Nunes C, Lima C, Cirilo E, Dantas F, Garcia A, Mendonça M (2012) On the proactive and interactive visualization for feature evolution comprehension: An industrial investigation. In: Proceedings of the 34th International Conference on Software Engineering (ICSE). IEEE Press, Piscataway. pp 1044–1053
Olbrich SM, Cruzes DS, Sjøberg DIK (2010) Are all code smells harmful? a study of god classes and brain classes in the evolution of three open source systems. In: Proc. of the 26th International Conference on Software Maintenance (ICSM). IEEE Computer Society, Washington, DC. pp 1–10
Padilha J, Figueiredo E, Sant'Anna C, Garcia A (2013) Detecting god methods with concern metrics: An exploratory study. In: Proc. of the 7th Latin-American Workshop on Aspect-Oriented Software Development (LA-WASP), co-allocated with CBSoft. ACM, New York
Palomba F, Bavota G, Penta MD, Oliveto R, Lucia AD (2014) Do they really smell bad? a study on developers' perception of bad code smells. In: Proc. of the 30th IEEE International Conference on Software Maintenance and Evolution (ICSME). IEEE Computer Society, Washington, DC
Parnin C, Görg C, Nnadi O (2008) A catalogue of lightweight visualizations to support code smell inspection. In: Proc. of the 4th international Smposium on Software visualization (SOFTVIS). ACM, New York. pp 77–86
Powers DMW (2012) The problem with kappa. In: Proc. of the 13th Conference of the European Chapter of the Association for Computational Linguistics (EACL). Association for Computational Linguistics, Stroudsburg. pp 345–355

Price BA, Baecker RM, Small IS (1998) An introduction to software visualization. In: Stasko J, Dominique J, Brown M, Price B (eds). Software Visualization. MIT Press, London. pp 4–26
Rapu D, Ducasse S, Girba T, Marinescu R (2004) Using history information to improve design flaws detection. In: Proc. of 8th European Conference on Software Maintenance and Reengineering (CSRM). IEEE Computer Society, Washington, DC. pp 223–232
Riel AJ (1996) Object-Oriented Design Heuristics. 1st edn. Addison-Wesley Longman Publishing Co., Inc., Boston
Romano J, Kromrey J, Coraggio J, Skowronek J (2006) Appropriate statistics for ordinal level data: Should we really be using t-test and Cohen'sd for evaluating group differences on the NSSE and other surveys? In: In annual meeting of the Florida Association of Institutional Research. pp 1–3
Santos JA, de Mendonça MG, dos Santos CP, Novais RL (2014) The problem of conceptualization in god class detection: agreement, strategies and decision drivers. J Softw Eng Res Dev (JSERD) 2(11):1–33
Santos JA, Mendonça M (2014) Identifying strategies on god class detection in two controlled experiments. In: Proc. of the 26th International Conference on Software Engineering and Knowledge Engineering (SEKE). Knowledge Systems Institute Graduate School, Skokie. pp 244–249
Santos JA, Mendonça M (2015) Exploring decision drivers on god class detection in three controlled experiments. In: Proc. of the 30th ACM/SIGAPP Symposium On Applied Computing (SAC). ACM, New York. pp 1–8
Santos JA, Mendonça M, Silva C (2013) An exploratory study to investigate the impact of conceptualization in god class detection. In: Proc. of the 17th International Conference on Evaluation and Assessment in Software Engineering (EASE). ACM, New York. pp 48–59
Schumacher J, Zazworka N, Shull F, Seaman C, Shaw M (2010) Building empirical support for automated code smell detection. In: Proc. of the 4th International Symposium on Empirical Software Engineering and Measurement (ESEM). ACM, New York. pp 1–10
Simon F, Steinbruckner F, Lewerentz C (2001) Metrics based refactoring. In: Proc. of 5th European Conference on Software Maintenance and Reengineering (CSMR). IEEE Computer Society, Washington, DC. pp 30–38
Sjøberg D, Yamashita A, Anda B, Mockus A, Dyba T (2013) Quantifying the effect of code smells on maintenance effort. IEEE Trans Softw Eng 39(8):1144–1156
Van Emden E, Moonen L (2002) Java quality assurance by detecting code smells. In: Proc. of the 9th Working Conference on Reverse Engineering (WCRE). IEEE Computer Society, Washington, DC. pp 97–106
Vinson NG, Singer J (2008) A practical guide to ethical research involving humans. In: Shull F, Singer J, DIK Søberg (eds). Guide to Advanced Empirical Software Engineering. Springer, London. pp 229–256
Whitehurst GJ (1984) Interrater agreement for journal manuscript review. Am Psychol 39(1):22–28
Wohlin C, Runeson P, Höst M, Ohlsson M, Regnell B, Wesslén A (2012) Experimentation in Software Engineering. Springer Berlin Heidelberg, Heidelberg
Yamashita A, Counsell S (2013) Code smells as system-level indicators of maintainability: An empirical study. J Syst Softw 86(10):2639–2653
Yamashita A, Moonen L (2013) Exploring the impact of inter-smell relations on software maintainability: An empirical study. In: Proc. of the 35th International Conference on Software Engineering (ICSE). IEEE Press, Piscataway. pp 682–691
Zhang M, Hall T, Baddoo N (2011) Code bad smells: A review of current knowledge. Softw Maint Evol Res Pract 23(3):179–202