



**Roberto Felício de Oliveira**

**To collaborate or not to collaborate? Improving  
the identification of code smells**

**Tese de Doutorado**

Thesis presented to the Programa de Pós-graduação em Informática of PUC-Rio in partial fulfillment of the requirements for the degree of Doutor em Ciências - Informática.

Advisor : Prof. Carlos José Pereira de Lucena

Co-advisor: Prof. Alessandro Fabricio Garcia

Rio de Janeiro  
August 2017



**Roberto Felício de Oliveira**

**To collaborate or not to collaborate? Improving  
the identification of code smells**

Thesis presented to the Programa de Pós-graduação em Informática of PUC-Rio in partial fulfillment of the requirements for the degree of Doutor em Ciências - Informática. Approved by the undersigned Examination Committee.

**Prof. Carlos José Pereira de Lucena**

Advisor

Departamento de Informática – PUC-Rio

**Prof. Alessandro Fabricio Garcia**

Co-advisor

Departamento de Informática – PUC-Rio

**Prof. Marcos Kalinowski**

UFF

**Prof.<sup>a</sup> Sabrina dos Santos Marczak**

PUCRS

**Prof. Arndt von Staa**

Departamento de Informática – PUC-Rio

**Prof.<sup>a</sup> Simone Diniz Junqueira Barbosa**

Departamento de Informática – PUC-Rio

**Prof. Márcio da Silveira Carvalho**

Vice Dean of Graduate Studies

Centro Técnico Científico – PUC-Rio

Rio de Janeiro, August the 21st, 2017

All rights reserved.

### **Roberto Felício de Oliveira**

Roberto Felício de Oliveira joined PUC-Rio as a Ph.D. student on Software Engineering in the Informatics Department in 2013. He visited the University of Southern California (USC) – EUA in 2016 to work with Prof. Nenad Medvidovic's team. Roberto was also a collaborator in under-graduate courses at PUC-Rio. In addition, he has also been a collaborator with me and has participated in the writing of a number of research projects. Roberto obtained a MSc degree in Computer Science (early 2013) from the Federal University of Paraíba (UFPB) and a BSc. degree in Computer Science (2003) from Paulista University (UNIP). His main research interests are in the area of code smells, collaborative practices, software product lines, and software maintenance.

#### Bibliographic data

Oliveira, Roberto Felício de

To collaborate or not to collaborate? Improving the identification of code smells / Roberto Felício de Oliveira; advisor: Carlos José Pereira de Lucena; co-advisor: Alessandro Fabricio Garcia. - 2017.

143 f: il. color. ; 30 cm

Tese (doutorado) - Pontifícia Universidade Católica do Rio de Janeiro, Departamento de Informática, 2017.

Inclui bibliografia

1. Informática – Teses. 2. Anomalia de Código;. 3. Identificação de Anomalia de Código;. 4. Identificação Colaborativa de Anomalia;. 5. Identificação Individual de Anomalia;. 6. Engenharia de Software Experimental;. I. Lucena, Carlos José Pereira. II. Garcia, Alessandro Fabricio. III. Pontifícia Universidade Católica do Rio de Janeiro. Departamento de Informática. IV. Título.

CDD: 004

## Acknowledgments

First and foremost I want to thank God for being by my side at all times. My courage to move on came from my faith in Him.

I would like to thank my beloved wife, Givanilde de Assis dos Santos Oliveira, who supported me during the realization of this work. There are no words to describe her love, patience and kindness. She unconditionally supported me in easy and hard times. I will be always grateful to my wife Givanilde Oliveira.

My deepest gratitude goes to my whole family: my father Felício V. de Oliveira (in memorian), my sisters Rose M. de Oliveira, Clarice C. de Oliveira and Regina A. de Oliveira, who always supported me. A special thank goes to my mom, Adelia M. S. de Oliveira, who was responsible for giving me the best opportunities. Without her I would not have come this far.

My deep gratitude goes to my advisor and friend Prof. Carlos Lucena for the continuous support of my Ph.D. study. I am indebted for his seemingly endless expertise, guidance, and patience during the last four years.

Special thanks go to my co-advisor Alessandro Garcia. I do not have words to express all my admiration and gratitude. His patience, energy and good will are incredible. He contributed significantly to my professional growth, providing the means for me to exceed my limits. From the beginning he was my guru, and a great friend. His knowledge was essential at every moment. We had great time during this adventure! Thank you for everything!

I would like to express my gratitude to the members of my thesis defense, Alberto Barbosa Raposo, Arndt von Staa, Marcos Kalinowski, Sabrina dos Santos Marczak, Simone Diniz Junqueira Barbosa, and Viviane Torres da Silva.

My sincere thanks also goes to Prof. Nenad Medvidovic, who provided me opportunities to join his team, and who gave access to his laboratories, research facilities, and to an indescribable life experience abroad. Without the team's precious support it would not be possible to conduct this research.

I thank all the professors from PUC-Rio for their contribution to my education. In particular, I would like to thank Simone Barbosa.

My sincere thanks also goes to professors Balduino Neto, Jaejoon Lee, Marcos Kalinowski, Pierre Bommel, Rafael Prikladnicki, Sabrina Marczak, Tayana Conte and Vaninha Vieira for their valuable insights.

I extend my gratitude to my colleagues and my friends from the OPUS Research Group and from the Software Engineering Laboratory. In particular, I would like to thank Danyllo Wagner, Diego Cedrim, Eduardo Fernandes, Isabella Vieira, Leonardo Sousa, Rafael de Mello and Willian Oizumi, for all their help and fellowship.



I am grateful for the time spent with flatmates, backpacking buddies, and new friends from Brazil, and USA. A special thanks to my new and old friends of a lifetime Adriana Lopes, Anderson Oliveira, Anderson Uchôa, Alexander López, Andrew Diniz da Costa, Ana Carla, Anne Benedicte Agbach, Arman Shahbazian, Bruno Cafeo, Bernardo Estácio, Carolina Valadares, Chrystinne Fernandes, Daniel Link, Davy Baia, Duc Le, Edson Oliveira, Francisco Cunha, João Neves, Joe Murray, Luciano Sampaio, Marx Viana, Natasha Valentim, Nathalia Nascimento, and Yixue Zhao and to the volunteers that participated in my studies. Particular thanks to Andre Lucena, Soeli Fiorini and Vera Menezes.

My sincere thanks the administrative staff of the Department of Informatics (DI) at PUC-Rio, specially Alex Alves, Cosme Leal and Regina Zanon.

I would like to also thank my colleagues at Universidade Estadual de Goiás - Câmpus-Posse by the contribution to make my participation in the doctoral program possible.

Finally, I gratefully acknowledge the Capes Foundation, Ministry of Education of Brazil through the PGCI/CAPES (No. 060/15) and PUC-Rio for financial support at different stages of this thesis.

## Abstract

Oliveira, Roberto Felício; Lucena, Carlos José Pereira (Advisor); Garcia, Alessandro Fabricio (Co-Advisor). **To collaborate or not to collaborate? Improving the identification of code smells**. Rio de Janeiro, 2017. 143p. Tese de doutorado – Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

Code smells are anomalous code structures which often indicate maintenance problems in software systems. The identification of code smells is required to reveal code elements, such as classes and methods, that are poorly structured. Some examples of code smell types perceived as critical by developers include God Classes and Feature Envy. However, the individual smell identification, which is performed by a single developer, may be ineffective. Several studies have reported limitations of individual smell identification. For instance, the smell identification usually requires an in-depth understanding of multiple elements scattered in a program, and each of these elements is better understood by a different developer. As a consequence, a single developer often struggles and to find to confirm or refute a code smell suspect. Collaborative smell identification, which is performed together by two or more collaborators, has the potential to address this problem. However, there is little empirical evidence on the effectiveness of collaborative smell identification. In this thesis, we addressed the aforementioned limitations as follows. First, we conducted empirical studies aimed at understanding the effectiveness of both collaborative and individual smell identification. We computed and compared the effectiveness of collaborators and single developers based on the number of correctly identified code smells. We conducted these studies in both industry's companies and research laboratories with 67 developers, including novice and professional developers. Second, we defined some influential factors on the effectiveness of collaborative smell identification, such as the smell granularity. Third, we revealed and characterized some collaborative activities which improve the developers' effectiveness for identifying code smells. Fourth, we also characterized opportunities for further improving the effectiveness of certain collaborative activities. Our results suggest that collaborators are more effective than single developers in: (i) both professional and academic settings, and (ii) identifying a wide range of code smell types.

## Keywords

Code Smell; Identification of Code Smell; Collaborative Smell Identification; Individual Smell Identification; Experimental Software Engineering;

## Resumo

Oliveira, Roberto Felício; Lucena, Carlos José Pereira; Garcia, Alessandro Fabricio. **Colaborar ou não Colaborar? Melhorando a Identificação de Anomalias de Código**. Rio de Janeiro, 2017. 143p. Tese de Doutorado – Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

Anomalias de código são estruturas anômalas de código que podem indicar problemas de manutenção. A identificação de anomalias é necessária para revelar elementos de código mal estruturados, tais como classes e métodos. Porém, a identificação individual de anomalias, realizada por um único desenvolvedor, pode ser ineficaz. Estudos reportam limitações da identificação individual de anomalias. Por exemplo, a identificação de anomalias requer uma compreensão profunda de múltiplos elementos de um programa, e cada elemento é melhor entendido por um desenvolvedor diferente. Logo, um desenvolvedor isolado frequentemente tem dificuldades para encontrar, confirmar e refutar uma suspeita de anomalia. Identificação colaborativa de anomalias, que é realizada em conjunto por dois ou mais colaboradores, tem o potencial para resolver esse problema. Porém, há pouca evidência empírica sobre a eficácia da identificação colaborativa de anomalias. Nesta tese, nós conduzimos estudos empíricos para entender a eficácia da identificação individual e colaborativa de anomalias. Computamos e comparamos a eficácia de colaboradores e desenvolvedores isolados com base no número de anomalias identificadas corretamente. Conduzimos tais estudos em empresas e laboratórios de pesquisa, totalizando 67 desenvolvedores, incluindo desenvolvedores novatos e experientes. Também definimos alguns fatores de influência sobre a eficácia da identificação colaborativa de anomalias, tais como a granularidade da anomalia. Revelamos e caracterizamos algumas atividades colaborativas que melhoram a eficácia dos desenvolvedores na identificação de anomalias. Finalmente, identificamos oportunidades para melhorar certas atividades colaborativas. Nossos resultados sugerem que colaboradores são significativamente mais eficazes que desenvolvedores isolados, tanto desenvolvedores novatos quanto experientes. Concluimos que colaborar é vantajoso para melhorar a identificação de uma vasta gama de tipos de anomalia.

## Palavras-chave

Anomalia de Código; Identificação de Anomalia de Código; Identificação Colaborativa de Anomalia; Identificação Individual de Anomalia; Engenharia de Software Experimental;

# Table of Contents

1	Introduction	14
1.1	Collaborative Smell Identification: A Motivating Example	15
1.2	Problem Statement	17
1.3	Goals and Research Questions	19
1.4	Research Methodology	20
1.5	Contributions	22
1.6	Outline	23
2	Background and Related Work	24
2.1	Basic Concepts	25
2.2	Studies on Code Smells	28
2.3	Studies on Developers' Collaboration	30
2.4	Limited Empirical Knowledge of Collaborative Smell Identification	32
2.5	Summary	33
3	Individual vs Collaborative Smell Identification: A Comparative Evaluation	34
3.1	Study Settings	35
3.1.1	Research Goal	35
3.1.2	Characterization of the Subjects	37
3.1.3	Target Software Systems and Data Sources	39
3.1.4	Data Analysis Procedures	40
3.1.5	Experiment Steps	42
3.2	Effectiveness of Collaborative Smell Identification	44
3.2.1	Analysis of Distribution for Precision and Recall	44
3.2.2	Comparing Precision and Recall of Collaborators and Single Developers	47
3.3	Threats to Validity	49
3.4	Summary	50
4	Understanding the Effectiveness of Collaborative Smell Identification	52
4.1	Study Settings	53
4.1.1	Research Goals	54
4.1.2	Characterization of the Subjects	55
4.1.3	Target Software Systems and Data Sources	57
4.1.4	Data Analysis Procedures	59
4.1.5	Experiment Steps	60
4.2	Influential Factors and Collaborative Activities	63
4.2.1	Influential Factors on the Identification of Code Smells	63
4.2.2	Collaborative Activities for Identifying Code Smells	67
4.3	Threats to validity	72
4.4	Summary	73
5	An Industrial Multi-Case Study on Collaborative Smell Identification	75
5.1	Study Settings	77
5.1.1	Goal and Research Questions	77

5.1.2	Target Software Development Organizations	78
5.1.3	Data Sources	80
5.1.4	Data Analysis Procedures	81
5.1.5	Steps of the Study Execution	82
5.2	Effectiveness of Collaborative Smell Identification in Industry	84
5.3	Improving Collaborative Smell Identification	87
5.4	Threats to Validity	91
5.5	Summary	92
6	Final Considerations	<b>94</b>
6.1	Main Findings	95
6.2	Recommendations	96
6.3	Future Work	97
	Bibliography	<b>99</b>
	APPENDIX	<b>109</b>
A	Appendix A - Consent Form	<b>110</b>
B	Appendix B - Characterization Questionnaire of Chapter 3	<b>112</b>
C	Appendix C - Characterization Questionnaire of Chapter 4	<b>115</b>
D	Appendix D - Characterization Questionnaire of Chapter 5	<b>118</b>
E	Appendix E - Code Smell Report Questionnaire	<b>121</b>
F	Appendix F - Follow-Up Questionnaire of Chapter 3	<b>123</b>
G	Appendix G - Follow-Up Questionnaire of Chapter 4	<b>126</b>
H	Appendix H - Follow-Up Questionnaire of Chapter 5	<b>129</b>
I	Appendix I - Organization Characterization	<b>132</b>
J	Appendix J - Support Material for Training	<b>135</b>

## List of Figures

Figure 1.1	Exchange of knowledge between collaborators	16
Figure 1.2	Phases of the research methodology	22
Figure 2.1	Code smells identification	26
Figure 2.2	Empirical study concepts of the thesis	28
Figure 3.1	The experimental design	42
Figure 3.2	Distribution of precision for developers	45
Figure 3.3	Distribution of recall for developers	46
Figure 3.4	Precision of collaborators and single developers per subject	47
Figure 3.5	Recall of collaborators and single developers per subject	48
Figure 4.1	Study steps	61
Figure 4.2	Average number of identified code smells per smell type	65
Figure 4.3	Activities on the identification of code smells	68
Figure 5.1	The experimental design	83

## List of Tables

Table 1.1	List of specific research questions	20
Table 1.2	Publications that contributed to this thesis	21
Table 1.3	Indirect publications	21
Table 2.1	Smell types considered in this thesis	27
Table 3.1	Study hypotheses derived from <i>RQ</i>	36
Table 3.2	Characterization of subjects	38
Table 3.3	Knowledge categorization about topics	38
Table 3.4	Comparison of precision and recall per working experience	46
Table 4.1	Hypotheses of <i>RQ</i> <sub>1</sub>	55
Table 4.2	Characterization of subjects	57
Table 4.3	Knowledge categorization about Topics	58
Table 4.4	General data about each software system	58
Table 4.5	Subject arrangement	63
Table 4.6	Average number of identified code smells	64
Table 5.1	Characteristics per target organization	78
Table 5.2	Subject characterization of CS1 and CS2	79
Table 5.3	Subject arrangement	84
Table 5.4	Precision and recall of single developers in CS1	85
Table 5.5	Precision and recall for collaborators in CS1	85
Table 5.6	Precision and recall for collaborators in CS2	85
Table 5.7	Precision and recall for single developers in CS2	86
Table 6.1	Summary of study findings of this thesis	95

## List of Abbreviations

CAPES – Coordination for the Improvement of Higher Education Personnel

CS – Computer Science

DI – Department of Informatics

OSS – Open Source Software

FN – False Negative

FP – False Positive

LOC – Lines of Code

PP – Pair Programming

PGCI – General Program for International Cooperation

PUC-RIO – Pontifical Catholic University of Rio de Janeiro

RQ – Research Question

SE – Software Engineering

SRQ – Specific Research Question

SS – Secondary Studies

TP – True Positive



*“No set of metrics rivals informed human intuition.”*

**Fowler**, *Refactoring: Improving the Design of Existing Code*, 1999.

# 1

## Introduction

Software systems evolve over time and, as a consequence, the source code is frequently changed (12). However, successive changes in the source code may lead to maintenance problems (68), such as the decay of the code structural quality (39). Developers may reveal several maintenance problems through anomalous code structures called code smells (29, 91). Examples of smell types are *God Class* and *Dispersed Coupling* (29). A single software system is likely to be affected by several code smells in different source code elements, such as classes and methods (33, 43). Certain code smells eventually make difficult to understand and change the affected code elements, which suggest the need to identify and eliminate these code smells whenever possible (1, 57, 58).

The identification of code smells consists of searching for anomalous code structures that potentially indicate maintenance problems in a software system (45). Code elements represent basic units of the software system decomposition, such as classes or methods (30, 37). Regardless of the code element affected by a code smell, its identification is more difficult than it is usually assumed or advertised (27, 63). In fact, the identification of code smells in practice consists of three basic steps, as follows (63, 65). First, developers identify potential code smell suspects in the source code. Second, developers inspect each code smell suspect. Third, developers either confirm or refute the code smell suspect as a true occurrence of a code smell in the software system.

Several industrial and academic tools aim at supporting the identification of code smell suspects (50, 53, 84). However, the inspection of a code smell suspect, as its confirmation or refutation, are naturally subjective, which requires the engagement of a developer. Despite the variety of available tools, previous work observe relevant shortcomings of using these tools, such as their usual low precision (11, 24). A low precision induces the misidentification of code smells and hinders the confirmation of code smell suspects by developers. In summary, even with tool support for its first step, smell identification remains challenging for developers.

Previous studies observe that developers usually identify code smells individually, which we refer as *single developers*, in an *ad-hoc* manner (66, 69). In addition, other studies observe that the identification of code smells may

become difficult or error-prone when performed by single developers (17, 63). Overall, there is empirical evidence that the identification of code smells by single developers has shortcomings that should be addressed. Thus, due to the subjectiveness of the identification of code smells and low precision of detection tools, developers face several difficulties that could be reduced or mitigated through the collaboration among developers (60).

## 1.1

### Collaborative Smell Identification: A Motivating Example

The inspection of each code smell suspect often requires an in-depth understanding of several code elements of the software system. For instance, to confirm or refute the occurrence of a *God Class* smell type (29), the developer has to understand whether the code smell suspect actually concentrates several responsibilities which should be implemented in multiple classes instead of a single class. The notion of centralizing multiple non-cohesive responsibilities is what characterizes a God Class. Thus, the developer requires knowledge about various responsibilities of the software system and what responsibilities are implemented in each class, including the God Class. However, this knowledge is sometimes difficult to obtain as the developers may not have implemented all involved classes. This lack of knowledge may suggest the need for exchanging knowledge among developers of the software system.

Figure 1.1 illustrates how developers working collaboratively, whom we refer to as *collaborators*, may benefit from the exchange of knowledge on the identification of code smells. The figure provides an example of knowledge exchange extracted from a real development scenario reported in our recent study (63). In the context of this thesis, the developer's *knowledge* may include information about the purpose of the software system, details on the code structure, or definitions of code smell types. In turn, *knowledge exchange* means sharing knowledge among developers, specially when identifying code smells together. We simplified the discussion among collaborators and changed the names of the developers due to privacy reasons.

Figure 1.1 represents two developers, namely Bill and Suzi, who have conducted a collaborative smell identification. The figure presents a part of the discussion among Bill and Suzi. We enumerate each statement of the discussion with a increasing number from 1 to 14, except for the numbers 6 and 13 that provide notes of the researchers. The increasing numeric order comprises the order of the statements. Along the discussion, we highlight key statements. A *key statement*, such as 1 and 3, means a statement that actually contributes to the identification of the code smell. We indicate the key statements to illustrate

when developers actually exchange useful knowledge for the identification of code smells. We provide an overview of the discussion as follows.

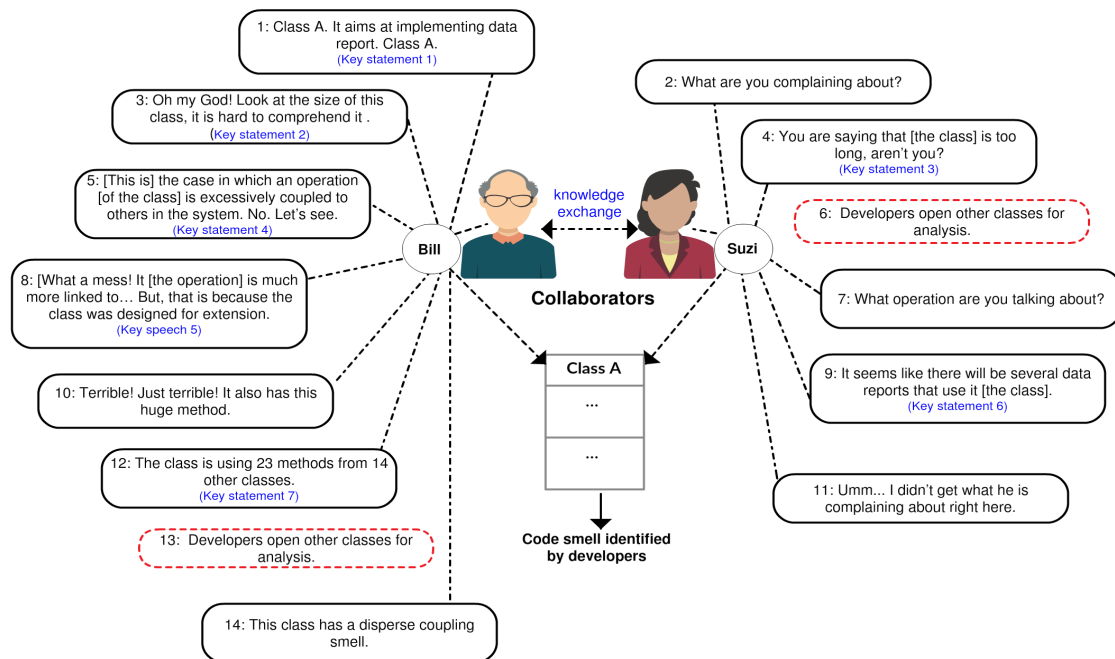


Figure 1.1: Exchange of knowledge between collaborators

Figure 1.1 suggests that Bill and Suzi work together to identify code smells in a system which they both know. In the beginning, Bill and Suzi had conflicting opinions about the occurrence of a code smell in Class A, which Bill assumes to have a *God Class* instance. First, Bill reflected upon the name of the class (*key statement 1*) to recall at least some of the responsibilities implemented by the class. Suzi then notices that Bill has tried to draw conclusions on the class responsibilities and asks Bill what he thinks about the class. Bill mentions the poor structural quality of Class A due to the class size (*key statement 2*), which implies the occurrence of a *God Class* code smell. However, Suzi does not feel confident to agree, since she assumes that Bill is taking into account only the size of the class (*key statement 3*). In fact, she does not believe that the class is affected by *God Class*.

After the aforementioned initial discussion between developers, Bill realizes that Suzi is right about considering only the class size to confirm the occurrence of a code smell. However, Bill is still convinced that the class contains a code smell with unknown smell type. Therefore, he tries to identify other indicators of the occurrence of a code smell in Class A. Consequently, Bill observes a method of Class A which has several dependencies with other classes of the software system (*key statement 4*). In order to avoid the misidentification of a code smell, Bill asks Suzi to help him with the inspection of the

other classes with which the suspicious one has any dependencies. He asks for Suzi's help because she has implemented the other classes and, therefore, her knowledge may be useful to confirm the occurrence of the code smell.

As Bill inspect the other classes, he realizes that Class A has other methods which depend on other classes (*key statement 5*). Based on Suzi's knowledge of these classes (*key statement 6*), Bill becomes confident that Class A is coupled to other classes because it depends on these classes to generate different data reports in the software system. Bill complements the discussion by counting how many methods Class A calls (*key statement 7*). At the end of the identification of code smells, the knowledge exchange leads both Bill and Suzi to confirm the occurrence of a *Dispersed Coupling* smell type in Class A. *Dispersed Coupling* occurs when a class depends just a little on several other classes of the software system (37).

In summary, the example illustrated in Figure 1.1 suggests that the knowledge exchange can be essential to avoid recurring omissions or mistakes on the identification of code smells performed by single developers. In fact, collaborators were able to minimize the misidentification of code smells via discussion, which would not be possible if the developers had worked in isolation. In addition, collaborators have improved their confidence when confirming or refuting the code smell suspect. Thus, collaborators are potentially more effective than single developers on the identification of code smells. However, there is no empirical knowledge if this superiority applies only to specific scenarios or to the confirmation or refutation of code smells in general. It might be that case that collaborative smell identification makes difficult confirming or refuting smells in certain circumstances.

## 1.2

### Problem Statement

Section 1.1 presented a motivating example of the collaborative smell identification. This example showed the knowledge exchange may be essential to the effectiveness of smell identification. In fact, the aforementioned example suggests that the collaborators actually benefit of discussions when identifying a code smell. Thus, the collaborative smell identification emerges as a possible way to improve the effectiveness of developers otherwise working in isolation, which has been shown to be limited for several reasons, such as the inherent difficulty to confirm or refute a code smell suspect (27, 44, 55, 63).

Although the collaborative smell identification potentially improves the developers' effectiveness, it is rarely applied by organizations due to the following reasons (43, 60, 63). First, there is limited knowledge of how to

conduct collaborative smell identification (63). Second, due to the limited knowledge, organizations assume that collaborative smell identification does not worth when compare to individual smell identification (43). Third, it remains unclear to what extent the collaborative smell identification improves the effectiveness of code smell identification performed by developers (60).

Chapter 2 details the aforementioned limitations. In summary, the literature suggests that the organizations need a complete body of knowledge regarding collaborative smell identification, in order to decide to what extent they may benefit from collaborative smell identification (43, 60, 63). Thus, all aforementioned problems are relevant, which leads us to the conduction of empirical studies focused on each problem. The problems presented above can be captured by the following general problem.

**General Problem.** To what extent collaborative smell identification is worthwhile remains unknown.

From our general problem, we have derived three specific problems which are addressed in this thesis. Each specific problem investigates the effectiveness of collaborative smell identification under a different viewpoint. We introduce each specific problem of the thesis as follows.

**Effectiveness of collaborative smell identification.** Section 1.1 raises the hypothesis that collaborators may be more effective than single developers on the identification of code smells. However, as aforementioned, one reason for organizations not adopting collaborative smell identification is that there is limited empirical knowledge of to what extent collaborators are actually more effective than single developers. In order to address this limitation, this thesis presents a series of empirical studies conducted in different settings, including real software development organizations. We aim at understanding the differences governing the effectiveness of collaborators and single developers.

**Problem 1.** The effectiveness of collaborative smell identification is unknown.

**Influential factors on the effectiveness of collaborative smell identification.** By assessing the effectiveness of collaborators and single developers on the identification of code smells, we provide the first insights on the benefits of the collaborative smell identification. However, it may not suffice for convincing organizations to adopt collaborative smell identification. Thus, similarly to previous work (2), we define factors such as the characteristics of software systems, development teams, or organizations that somehow influence the

collaborative smell identification. For instance, does the smell granularity influence the effectiveness of smell identification? Or, as suggested in the example of Section 1.1, does the knowledge of developers on the system under inspection influence the smell identification? In summary, we aim at answering these and other questions related to influential factors on the effectiveness of collaborative smell identification.

**Problem 2.** The influential factors on the effectiveness of collaborative smell identification are unknown.

**Activities of collaborative smell identification.** Some organizations reportedly do not adopt the collaborative smell identification as they know little about how to conduct it (63). Thus, after identifying the influential factors on the effectiveness of collaborative smell identification, we aim at understanding what collaborative activities are influenced by these factors. We refer as *activity* an action of developers with a specific goal during the identification of code smells. An example of activity is the set of actions for confirming or refuting a code smell suspect. In this thesis, we aim at identifying the activities that are mostly collaborative. We refer as *collaborative activity* an activity that should be performed by collaborators to improve the effectiveness of smell identification. In addition, we aim at identifying the collaborative activities that could be further improved with the aim at further increasing effectiveness.

**Problem 3.** The collaborative activities which compose the identification of code smells are unknown.

### 1.3

#### Goals and Research Questions

After observing there is limited empirical knowledge of collaborative smell identification (Section 1.2), we designed the study goal (89) of this thesis as follows: *analyze* the collaborative smell identification; *for the purpose of* comparing it with individual smell identification; *with respect to* the developers' effectiveness; *from the point of view of* developers with different backgrounds and levels of working experience; *in the context of* Brazilian project settings. In summary, we address a general research question described as follows.

**General Research Question:** To what extent the collaborative smell identification improves the developers' effectiveness?

In the context this thesis, we consider the developer's effectiveness as the capability of a developers to correctly confirm or refute the existence of a code smell, based on a code smell suspect obtained through an automated or a manual inspection. To deeply investigate our general research question, we split it into four specific research questions ( $SRQ_s$ ) presented in Table 1.1.

Table 1.1: List of specific research questions

$SRQ_s$	Description
$SRQ_1$	Does the collaborative smell identification improve the developers' effectiveness when compared to the individual smell identification?
$SRQ_2$	What influential factors contribute to the developers' effectiveness on collaborative smell identification?
$SRQ_3$	What collaborative activities contribute to the developers' effectiveness on smell identification?
$SRQ_4$	Are there opportunities for improving collaborative activities of smell identification?

With  $SRQ_1$  we assess whether collaborators are more effective than single developers on the identification of code smells. We conducted a controlled experiment with 26 novice and professional developers unfamiliar with the analyzed software systems. With  $SRQ_2$  we investigate the influential factors on the effectiveness of collaborative smell identification. We conducted a controlled experiment with 28 novice developers also unfamiliar with the analyzed systems. With  $SRQ_3$  we reveal the collaborative activities typically involved in the identification of code smells. We conducted a qualitative analysis on how developers identify code smells, based on the data of  $SRQ_2$ . Finally, with  $SRQ_4$  we reveal opportunities for improving collaborative activities of smell identification. We conducted an industrial case study with 13 professional developers familiar with the inspected systems.

Table 1.2 lists the publications that contributed to this thesis. The first and second columns reference and relate each publication with our  $SRQ_s$ . In addition, Table 1.3 lists the publications generated during this doctoral research and which have an indirect relationship with this thesis.

## 1.4 Research Methodology

Empirical studies have been increasingly conducted by researchers aimed at assessing the effectiveness of techniques and methodologies in software engineering (49, 89). There are different empirical studies with varied purposes, such as evaluating a novel technique or understand the state of the art in a given research topic. This thesis combines different empirical studies to address our specific research questions (Table 1.1): literature review (Chapter 2), controlled experiment (60, 61, 62, 64), and case study (63).



Table 1.2: Publications that contributed to this thesis

Publication	SRQ <sub>s</sub>
<b>Oliveira, R.</b> ; Sousa, L.; Mello, R.; Valentim, N.; Lopes, A.; Conte, T., Garcia, A.; Oliveira, E. and Lucena, C. Collaborative Identification of Code Smells: A Multi-case Study. In: 39th ACM/IEEE International Conference on Software Engineering (ICSE), Software Engineering in Practice (SEIP) Track. p. 33–42, Buenos Aires, 2017.	SRQ <sub>1,3,4</sub>
<b>Oliveira, R.</b> ; Mello, R.; Garcia, A. and Lucena C. Evaluating the effectiveness of pair programming on the identification of code smells: An empirical study. In: Journal of Systems and Software (JSS), under minor review, 2017.	SRQ <sub>1</sub>
<b>Oliveira, R.</b> When more heads are better than one? Understanding and improving collaborative identification of code smells. In: 38th ACM/IEEE International Conference on Software Engineering (ICSE), Doctoral Symposium, p. 879–882, Austin, 2016.	ALL
<b>Oliveira, R.</b> ; Estacio, B.; Garcia, A.; Marczak, S.; Prikladnicki, R.; Kalinowski, M. and Lucena, C. Identifying code smells with collaborative practices: A controlled experiment. In: 10th Brazilian Symposium on Components, Architectures, and Reuse (SBCARS), p. 61–70, Maringá, Brazil, 2016.	SRQ <sub>1,2,3</sub>
<b>Oliveira, R.</b> ; Garcia, A.; Lucena, C. and Albuquerque, D. A Eficácia de pares na identificação de anomalias de código: Um experimento controlado. In: 11th Workshop on Software Modularity (WMod), p. 53–66, Maceió, Brazil, 2014.	SRQ <sub>1</sub>

Table 1.3: Indirect publications

Publication
Mello, R.; <b>Oliveira, R.</b> and Garcia, A. Investigating the influence of human factors on the identification of code smells. In: 11st ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM), p. 1–10, Toronto, Canada, 2017.
Estácio, B.; <b>Oliveira, R.</b> ; Marczak, S.; Kalinowski, M.; Garcia, A.; Prikladnicki, R. and Lucena, C. Evaluating collaborative practices in acquiring programming skills: Findings of a controlled experiment. In: 29th Brazilian Symposium on Software Engineering (SBES), p. 150–159, Belo Horizonte, Brazil, 2015.
Mello, R.; <b>Oliveira, R.</b> ; Sousa, L. and Garcia, A. Towards effective teams for the identification of code smells. In: 10th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE), co-located with 39th ICSE, p. 62–65, Buenos Aires, Argentina, 2017.

A *literature review* aims at collecting scientific evidence on a specific research topic, such as the collaborative smell identification. In turn, a *controlled experiment* focuses on understanding cause-effect relationships with randomly selected subjects (78). Controlled experiments usually involve novices and professional developers, which conduct the experimental tasks in a short time (78). A *case study* aims at investigating a phenomenon in real settings, even when the phenomenon is unclear (75). Both controlled experiments and case studies are known in the literature as *primary studies*.

Figure 1.2 illustrates the four phases of our research methodology. We discuss each phase in detail as follows.

**Phase 1** consisted of a literature review aimed at understanding the identification of code smells and collaboration practices in software engineering. As a result, we collected empirical evidence on both topics. Based on the results of this phase, we elaborated the terminology presented in Section 2.1. **Phase 2** consisted of conducting controlled experiments aimed at assessing the developers’ effectiveness on collaborative smell identification (60, 61, 62, 64). For this purpose, we compare the effectiveness of both collaborators and single developers that are unfamiliar with the inspected systems, which represents a common software development scenario. We then answer three specific research questions of this thesis: *SRQ<sub>1</sub>*, *SRQ<sub>2</sub>*, and *SRQ<sub>3</sub>*.

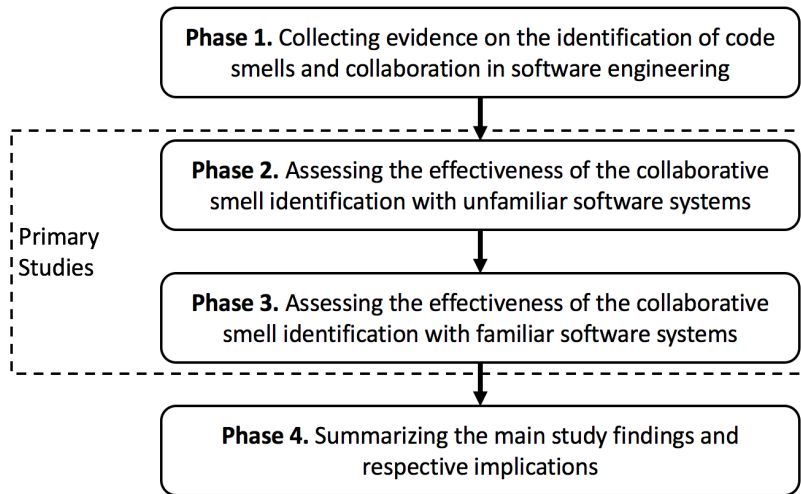


Figure 1.2: Phases of the research methodology

**Phase 3** consisted of a case study aimed at reinforcing the findings of Phases 1 and 2 with respect to collaborative smell identification. Differently of the previous phases, we focus on industrial settings and developers which are familiar with the software systems under analysis (63). We then answer the two last specific research questions:  $SRQ_3$  and  $SRQ_4$ . Finally, **Phase 4** consists of summarizing the study findings on collaborative smell identification. For this purpose, we revisit our empirical studies aimed at gathering the most relevant findings with the respective implications. Our goal was to provide a thesis overview, which concludes that collaborators are more effective than single developers on the identification of code smells.

## 1.5 Contributions

This thesis presents empirical studies aimed at assessing the effectiveness of collaborative smell identification. Based on the results obtained for each specific research questions, we draw some interesting conclusions as follows.

- The investigation of  $SRQ_1$  reveals that collaborators are usually more effective than single developers when identifying code smells. In fact, collaborators more easily identify certain smell types, such as *God Class* and *Dispersed Coupling*, when compared to single developers. Thus, organizations should consider adopting collaborative smell identification to improve their effectiveness.
- By investigating  $SRQ_2$ , we observed certain influential factors on collaborative smell identification. One influential factor that we have found is the smell type. In fact, our data suggest that the identification of

inter-class smell types, which require knowledge about multiple classes, may benefit from collaborative smell identification. Overall, the influential factors observed in our study may help organizations in carefully applying collaborative smell identification.

- By assessing  $SRQ_3$ , we characterized activities which compose the identification of code smells, performed by collaborators and single developers. There are many different actions, from deciding about the smell types which mostly concern the developers, to confirming or refuting a code smell suspect as an actual code smell. We also characterized the activities that improve the developers' effectiveness at most. By characterizing these activities, we provide preliminary insights on how organizations should implement collaborative smell identification.
- Finally, by investigating  $SRQ_4$ , we identified opportunities for improving collaborative activities. These opportunities aim at improving the developers' effectiveness on collaborative smell identification. For instance, developers often need to exchange knowledge to confirm or refute a code smell suspect in collaboration. However, we observed that developers require additional information during the collaborative smell identification, such as the evolution of the code elements towards the occurrence of code smell suspects, which is often not provided by existing tool support.

## 1.6 Outline

The remainder of this thesis is organized as follows. Chapter 2 provides background information to help understand the thesis. Chapter 3 presents an empirical study aimed at assessing the effectiveness of collaborative smell identification. Chapter 4 discusses an empirical study with the two focuses. First, we investigate the influential factors on collaborative smell identification. Second, we investigate collaborative activities which improve the developers' effectiveness on collaborative smell identification. Chapter 5 reinforces the study findings of Chapters 3 and 4 in industry settings. In addition, we investigate opportunities for improving the collaborative activities and, consequently, improve the collaborators' effectiveness. Finally, Chapter 6 concludes the thesis with a summary of our contributions and suggestions for future research.

## 2

## Background and Related Work

Literature reports that maintenance problems often affect software systems (43, 79). Code smells provide hints of certain maintenance problems (29, 91) that actually increase the effort to read, change, and fix the source code (84, 92). Aimed at supporting the identification of code smells, previous work assessed the effectiveness of individual smell identification, which is performed by single developers (17, 55). They usually observed limitations of individual smell identification (27, 55), which could be addressed through developers' collaboration. Thus, collaborative smell identification emerges as an opportunity to improve the identification of code smells (63). However, there is limited empirical evidence on the circumstances that collaborative smell identification is indeed effective.

To address the aforementioned limitation, this thesis presents empirical studies aimed at understanding the effectiveness of collaborative smell identification. This chapter provides background information aimed at supporting the comprehension of this thesis. We present the basic concepts which underlies all next chapters of this thesis. We also discuss related work in four parts, aimed at contextualizing our studies from different perspectives in the literature. First, we present studies that aimed at understanding the identification of code smells in general. Second, we present studies that aimed at assessing the effectiveness of individual smell identification. Third, we discuss previous work organized on collaboration practices of developers along different software engineering activities. Fourth, we discuss the lack of empirical evidence on collaborative smell identification.

The remainder of this chapter is organized as follows. Section 2.1 presents the basic concepts of the thesis. The other sections discuss related work as follows. Section 2.2 discusses identification of code smells in general. Section 2.3 overviews developers' collaboration along software engineering activities. Section 2.4 discusses the limited knowledge about the effectiveness of collaborative smell identification. Finally, Section 2.5 concludes this chapter and introduces the next chapter.

## 2.1

### Basic Concepts

This section presents the basic concepts which guide the understanding of the thesis. We introduce these concepts in two parts. First, we present the code smells identification, such as code smells, smell types, and both individual and collaborative smell identification. Second, we present concepts associated with the empirical studies of this thesis, such as effectiveness of smell identification, and influential factors and collaborative activities on smell identification. We introduce these concepts as follows.

**Code Smells Identification Concepts.** The code smell identification concepts include from the definition of code smells to the elementary terms regarding the code smell identification, which are fundamental for the understanding of this thesis. These concepts are reported in the Figure 2.1. The figure models the relationships among different concepts using on the Unified Modeling Language (UML) notation (14). Each rectangle represents a basic concept. In addition, each continuous line represents a hierarchical relationship between two concepts. This relationship means that the concept represented at the most above level is decomposed into the concepts at the level below. Finally, the dotted lines represent the relationships of dependency among multiple concepts. We discuss the figure in detail as follows.

A core basic concept presented in Figure 2.1 is *code element*. A code element consists of the basic decomposition unit of a software system (30). This decomposition unit is either a *class*, which is composed by *methods* and *attributes*, or a *method*, which is composed by *statements* (37). Certain code elements are also *code smell suspects*. A code smell suspect consists of a code element which is possibly affected by a *code smell*, but still not confirmed or refuted by developers as actually affected by the code smell. In turn, a code smell is an anomalous code structure which often indicates one or more maintenance problems in a software system (29, 91).

Regarding code smells, each instance of code smell has a *smell type*. Examples of smell types are *Dispersed Coupling*, which means a class with several dependencies with other classes of the system (29), and *Long Method*, which means a method with excessive length and complexity (29). In addition, each smell type has a different *smell granularity* classified as *intra-class smell* or *inter-class smell* (29, 46). *Intra-class smells* are anomalous code structures which affect a particular class of the system, such as *Long Method* and *Long Parameter List*. *Inter-class smells* are those which affect multiple classes together, such as *Feature Envy* and *Message Chain*.

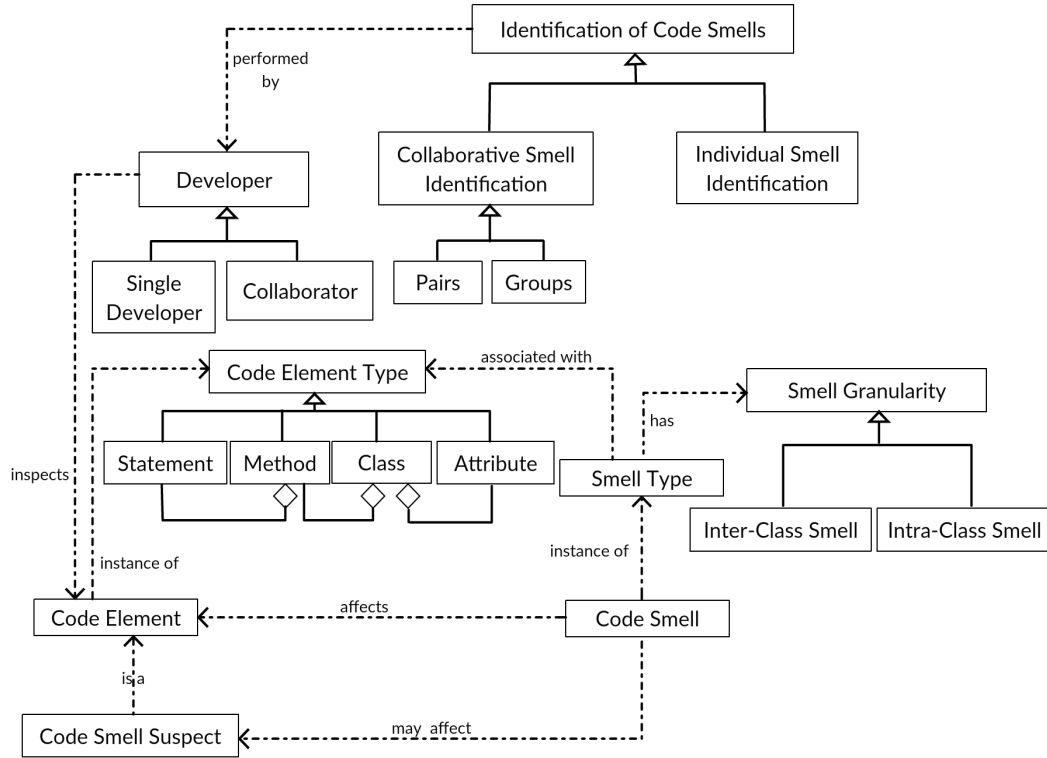


Figure 2.1: Code smells identification

Table 2.1 presents the smell types inspected in this thesis. The first column lists the smell types. The second column defines each smell type. The third column indicates which specific research question(s) (*SRQs*) considered the respective smell type, presented in each row. We do not consider all the smell types in all empirical studies (*SRQs*) of this thesis because a particular smell type may be not relevant to a software project. For instance, certain smell types rarely occur in certain projects. All smell types of Table 2.1 are defined in the literature (16, 29), are commonly investigated (24, 56, 92), may cause the decay of code structural quality (43). Finally, certain smell types may co-occur (67). For instance, a class affected by *God Class* may also be affected by *Long Method* and *Feature Envy*.

Another basic concept is *identification of code smells*, which has three steps: to identify code smell suspects in the source code; to inspect each code smell suspect; to confirm or refute each code smell suspect as a true occurrence of code smell (65). Developers usually perform the identification of code smells alone. The *individual smell identification* occurs whenever a *single developer* identifies code smells alone (55). The *collaborative smell identification* occurs whenever two or more developers identify code smells together. In this case, each developer is called *collaborator*. Collaborators may be allocated in *pairs* or *groups* with more than two developers.

Table 2.1: Smell types considered in this thesis

Intra-class smells		
Smell Type	Description	SRQ <sub>s</sub>
Complex class	Class whose methods have a high cyclomatic complexity	SRQ <sub>3</sub>
Large class	Class that is excessively long	SRQ <sub>2</sub>
Long method	Method with long length and complexity	ALL
Long parameter list	Method with several parameters, some of which unnecessary	SRQ <sub>3</sub>
Spaghetti code	Class with long methods that are excessively inter-dependent	SRQ <sub>3</sub>
Speculative generality	Abstract class with a few concrete classes using its methods	SRQ <sub>3</sub>
Inter-class smell		
Smell Type	Description	SRQ <sub>s</sub>
Data class	Class with fields, getter and setter methods only	SRQ <sub>2</sub>
Data clumps	Data often found together in classes or method calls	SRQ <sub>1</sub>
Duplicated code	Similar parts of code elements that occur in different classes	SRQ <sub>2,3</sub>
Feature envy	Method mostly concerned with another class than the one it belongs	SRQ <sub>2,3</sub>
God class	Class with long length and several implemented responsibilities	SRQ <sub>2,3</sub>
Lazy class	Small class with only a few methods and low complexity	SRQ <sub>2,3</sub>
Message chain	Several method calls to realize a single responsibility	SRQ <sub>1,3</sub>
Refused bequest	Child class that does not use the functionalities of its parent	SRQ <sub>3</sub>
Shotgun surgery	Changes in a class require changing several others	SRQ <sub>3</sub>

**Empirical Study Concepts.** The empirical study concepts consist of elementary terms which are fundamental for the understanding of our quantitative and qualitative studies. In summary, they regard concepts which we assess by conducting empirical studies. Figure 2.2 presents another part of the basic concepts which regards the empirical study concepts. Similarly to Figure 2.1, we represent these concepts with a notation inspired by UML (14). We discuss in detail each empirical study concept as follows.

A concern concept in our empirical studies is the *effectiveness of smell identification*. In short terms, effectiveness means to correctly identify code smells and avoid misidentified code smells. This thesis assesses the effectiveness of smell identification in two ways, i.e. we investigate such an effectiveness based on both quantitative and qualitative analyses. Both analyses are complementary, because each of them provides different viewpoints and findings about the developers' effectiveness on the identification of code smells. We discuss both analyses as follows.

In our quantitative analysis, we compute effectiveness through the *average number of identified code smells* and two largely used measures (6, 11, 24), namely *precision*, and *recall*. The average number of identified code smells is the total number of identified code smells divided by the total number of subjects involved in the identification of code smells. In turn, precision measures the correctness of the identified code smells. Finally, recall measures the completeness of the identified code smells with respect to all code smells which occur in a system (23). All these measures depend on *code smell reference lists*, which are built based on the report of one or more *smell detection tools* or a *manual smell identification*. A code smell reference list itemizes the code smells identified in the source code of a software system (25).

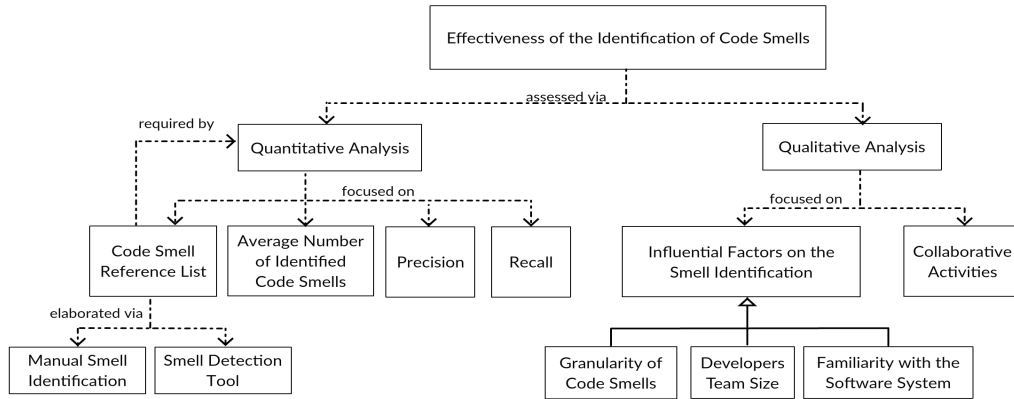


Figure 2.2: Empirical study concepts of the thesis

Regarding the qualitative analysis, we focus on the *factors* and *collaborative activities* which contribute to improve the developers' effectiveness on collaborative smell identification. Factors are characteristics related to software systems, development teams, or organizations which contribute to the effectiveness of the smell identification. In turn, activities are actions with specific goals and performed by developers during the identification of code smells, such as the ones performed to confirm or refute a code smell suspect.

Finally, regarding factors, our studies focus on three factors. First, the *smell granularity*, since different granularities are inherently difficult to inspect by single developers. Based on a previous work (80), we consider granularity as the code abstraction level affected by a code smell suspect: at class (general) or method (specific) level. Thus, collaboration may make such an inspection easier. Second, the *size of the developers team* which identify code smells, since the number of developers involved in the identification of code smells may positively or negatively affect the team's effectiveness. Third, the *familiarity with the software system*, which means that developers have some or knowledge of the analyzed systems. Because some developers know little about the system, collaboration could be helpful to improve the developers' effectiveness.

## 2.2

### Studies on Code Smells

Code smells are anomalous code structures that suggest maintenance problems (29), which may increase the maintenance effort (19, 58) and decrease the code comprehensibility (1). Studies assessed the side effects of code smells on structural quality (59, 34). Others show that code smells co-occur, which decay even more the code structural quality (90, 57). Finally, studies investigate the task of identifying code smells (54, 65). We discuss these studies as follows.



**Side Effects of Code Smells on Software Systems.** In order to assess to what extent code smells actually lead to maintenance problems, Olbrich, Cruzes and Sjöberg (2010) analyzed development historical data of open source software systems. Their results suggest that certain smell types, such as *God Class* and *Brain Class*, cause the decay of the code structural quality, which consequently makes difficult to maintain the source code. In turn, D'Ambros *et al.* (2010) assessed how code smells increase the effort spent by developers to fix problems in the source code of open source systems. As a result, the authors observe that, the higher the number of code smells, the higher is the developers' effort to fix problems in the source code. Thus, developers should identify and eliminate code smells whenever possible.

Regarding the changeability of the source code, previous studies investigated whether code smells lead to a high number of changes in the code elements affected by code smells (34, 42, 59). By assessing the historical data of open source software systems, they conclude that code smells actually lead to several changes in the affected code elements. As aforementioned, developers should identify and eliminate the code smells which affect the software systems. Consequently, developers could improve the code structural quality and increase the longevity of their systems (51, 77).

Abbes *et al.* (2011) assesses the effects of code smells on code comprehensibility, which means how easily developers could read and understand the source code (26). Their results suggest that, for open source systems, certain smell types such as *Blob* and *Spaghetti Code* (16) actually cause the decay of the code comprehensibility. That is, the anomalous code structures are often difficult to read and understand. Regarding code comprehensibility, our thesis aimed at making it easier for developers to characterize and identify smelly structures through developer collaboration. We hypothesize that such collaboration is likely to improve their effectiveness on smell identification.

Finally, recent studies provide empirical evidence that certain smell types often co-occur in the software system. Consequently, the co-occurrence of different code smells may increase the negative effects on each code smell the code structural quality (57, 86, 90). For instance, Oizumi *et al.* (2016) observe that the co-occurrence of code smells manifest in different ways, such as the so-called code smell agglomerations, and may involve multiple code elements together, from different parts of the source code. Consequently, the co-occurrence of code smells makes it difficult to identify and eliminate code smells, aimed at improving the code structural quality. In this context, this thesis aims at understanding how collaborative smell identification could ease the identification of code smells, which may affect multiple code elements.

**Studies of Code Smell Identification.** Section 2.1 discusses the three steps of the identification of code smells (65): to identify the code smell suspects; to inspect each suspect; and to either confirm or refute each suspect. Previous studies assess means to support each step (18, 72). For instance, Pietrzak and Walter (2006) suggests combining multiple characteristics of the source code to improve the effectiveness of the identification of code smells. Other studies propose minimizing the effort of developers when conducting each step (50, 53). However, there is still limited empirical knowledge about how these steps could be improved, which we address in this thesis by understanding how collaborative smell identification could benefit each step.

We also discuss two ways to identify code smells: the manual smell identification (54), which naturally requires the engagement of developers; and the use of smell detection tools (50), which partially automates the identification of code smells, but also requires developer's engagement. Previous studies investigate whether the smell detection tools actually support the identification of code smells (11, 24, 28). For instance, Fernandes *et al.* (2016) observe that the detection results provided by tools vary significantly due to the diverse strategies adopted to identify code smells is still unclear. Thus, developers should not totally rely on the report of these tools, which requires the developers' engagement to confirm or refute the existence of a code smell suspect.

## 2.3

### Studies on Developers' Collaboration

As discussed in Section 2.2, literature has reported several shortcomings of smell identification, mostly when such an identification is performed by a single developer. In fact, there is empirical evidence that even highly skilled developers have limitations (88). Thus, the collaboration of developers emerges as a possibility to improve the developers' effectiveness on the identification of code smells. In fact, previous work suggests that the collaboration of developers is essential for improving their effectiveness on the development and maintenance of software systems (31). We call *collaboration* when two or more developers work together and exchange knowledge with a specific goal during the development or maintenance of a system (87).

Previous studies have investigated the collaboration of developers in software engineering (5, 8, 74, 83). They suggest that the collaboration in software engineering is effective to address the limitations of single developers. In fact, developers working as collaborators benefit in two ways. First, collaborators could avoid mistakes otherwise made by each single developer. Second, collaborators could improve their effectiveness in the system development

and the identification of code smells, for instance. However, by no means we should expect that it is obvious that collaboration (always) improves smell identification. Each smell has a simple structural pattern, which is likely to be easily identified by a single developer, in particular an experienced one. Unfortunately, there is limited knowledge of either adopting or assessing collaboration along the identification of code smells. Nevertheless, previous work investigates other applications, as discussed next.

**Collaboration in Code Review.** *Code review* means inspecting the software system to identify a wide range of problems which developers should address (74). Code review consists of allocating developers to inspect code elements aimed at identifying violations of organization recommendations, development standards, and general problems (5, 13). Organizations such as Google and Microsoft (7, 73) have adopted collaboration in their code review practices to improve the developers' effectiveness. Particularly, Sutherland and Venolia (2009) observe a major benefit of collaboration in code review. That is, the knowledge exchanged among developers actually improves the developers' effectiveness in code review, by supporting the understanding and the reasoning about the software system. However, they explored the use of developers' collaboration to identify widely-scoped structural problems (e.g. major architectural problems). They did not assess the role of collaboration in improving the identification of apparently simpler structure of code smells.

Organizations differently apply code review (5, 10, 73) regarding how developers collaborate during the inspection of code elements (47). One developer is usually responsible for implementing certain code elements, and others assess the quality of those code elements, aimed at verifying possible inconsistencies and problems, such as bugs (47, 48). Bugs are any mistakes made by developers during the software development and maintenance, which change the expected behavior of a software system (36). However, there is no understanding of whether these organizations can actually benefit of existing developers' collaboration to improve smell identification.

**Collaboration in Pair Programming.** *Pair Programming (PP)* consists of two developers working together on the same development task, such as adding code elements to the system or inspecting existing code elements (4). Usually, one developer called *driver* implements the code, and another developer called *observer* assists the driver by reviewing the code, for instance. Developers can switch roles (driver and observer). Whenever developers only inspect the code elements, both developers analyze the code as observers (22).

Previous work discuss the advantages of applying pair programming to improve the code structural quality (8, 82). For instance, they show that pair programming makes developers effective in identifying defects in code elements (70, 82). Previous work also observe that the knowledge exchange is essential for the developers' effectiveness in pair programming (32, 85). They show that pair programming requires continuous discussion between developers (85), aimed at supporting the identification of defects in the software system and the review of complex code elements (32). This thesis innovates by assessing the role of collaboration in pairs to either reduce smells prevailing in the code along programming sessions or improve smell identification in existing code.

## 2.4

### Limited Empirical Knowledge of Collaborative Smell Identification

As discussed in Section 2.3, the collaboration of developers has been applied and assessed in software engineering tasks. For instance, previous studies show that the code review may benefit from collaboration, since developers working together are able to reveal more precisely defects and maintenance problems in software systems (5, 13). In addition, we discuss in Section 2.2 the drawbacks of individual smell identification, which could benefit from developers' collaboration. However, there is limited evidence as to what extent the developer collaboration indeed improves their effectiveness.

Previous work shows that collaboration reduces the difficulty faced by developers to identify defects in software systems, which manifest in the source code (47, 48). Consequently, one could expect that the collaboration has potential to improve developers' effectiveness in identifying code smells, which also manifest in the source code of a system. However, we did not find studies aimed at investigating such potential. Moreover, studies show that the knowledge exchange supports developers in understanding the source code (83). Therefore, it would be interesting to explore whether collaborative smell identification helps developers in reasoning about code smells. Again, there is limited empirical knowledge about this issue.

In summary, although collaboration potentially improves the developers' effectiveness in identifying code smells, little is known about such improvement from the literature. Due the lack of knowledge, several organization do not consider applying collaboration in their development contexts. Thus, organizations need a complete body of knowledge regarding collaborative smell identification, in order to decide to what extent they may benefit from collaborative smell identification (43, 60, 63). Due to the aforementioned limitations, we conducted various empirical studies aimed at addressing this issue.

## 2.5

### Summary

This chapter provided the required background to support the understanding of this thesis. We presented our basic concepts, which defines the main terms used throughout the next thesis chapters. Besides the basic concepts, we discussed related work as follows. First, we presented empirical studies on the occurrence and side effects of code smells. Second, we discussed studies on the collaboration of developers in software engineering. Third, we discussed the limited empirical knowledge of collaborative smell identification. We discussed how this thesis assesses the effectiveness of collaborative smell identification.

Based on the discussion presented in this chapter, we claim that it is not clear whether developers' collaboration can indeed improve their effectiveness on smell identification. Thus, the next chapter provides the first insights on the effectiveness of collaborative smell identification. For this purpose, we conducted an empirical study aimed at understanding whether collaborators are more effective than single developers when identifying code smells.

## Individual vs Collaborative Smell Identification: A Comparative Evaluation

According to the literature, developers usually identify code smells individually (27, 44, 57). By assuming that, studies assess the effectiveness of individual smell identification and report drawbacks of this identification, such as the difficulty of developers to confirm or refute a code smell suspect (65). Since the identification of code smells is subjective and widely depends on the developer expertise, one should expect that developers identifying code smells collaboratively, the so-called *collaborators* (Section 2.1) are more effective than the ones identifying code smells individually, called *single developers*. Developers' collaboration may improve their effectiveness by improving smell identification correctness, i.e. without misidentifying smells.

Despite the potential of collaborative smell identification, there is limited empirical knowledge of its actual impact on developers' effectiveness. Moreover, most industrial and academic parties simply assume a single skilled developer is effective enough to identify the apparently simple structure of each smell. To address this lack of empirical knowledge, this chapter presents an empirical study aims at comparing the effectiveness of both collaborators and single developers. The study design is grounded in the terminology presented in Section 2.1. Additionally in this section, we discuss precision and recall, which we used to compute effectiveness. In summary, we address the first specific research question of this thesis ( $SRQ_1$  of Section 1.3), which states: *Does the collaborative smell identification improve the developers' effectiveness when compared to the individual smell identification?*

To answer  $SRQ_1$ , we conducted a controlled experiment with 26 developers as subjects. We classify the subjects by level of working experience, namely *novice developers* and *professional developers* (Section 3.1.2). We split the experiment into two sessions: one session with novice developers, and the other with professional developers. During each session, the subjects first perform the individual smell identification and, in the sequence, they perform the collaborative smell identification. This experiment aims at providing the first insights for organizations on to what extent the collaborative smell identification is worthwhile. Consequently, organizations could save maintenance effort

by identifying code smells which are actual problems in software systems.

As a result of this empirical study, we observe that collaborators are actually more effective than single developers in identifying code smells. This observation relies on the following study findings.

- Collaborators are more effective than single developers on the identification of code smells, regardless of their working experience. In fact, both precision and recall have improved with collaborative smell identification.
- Collaborators benefit from information exchange during the identification of code smells. Consequently, they are able to correctly identify several code smells and minimize the number of misidentified code smells.

The empirical study reported in this chapter was submitted for review to the Journal of Systems and Software (64)<sup>1</sup>. The remainder of this chapter is organized as follows. Section 3.1 describes the study settings, including the study goal and research questions. Section 3.2 presents the results of our empirical study regarding the effectiveness of collaborative smell identification. Section 3.3 discusses threats to the study validity. Section 3.4 summarizes this chapter and introduces the following chapter.

### 3.1 Study Settings

This section describes the settings of our empirical study aimed at understanding whether collaborators are more effective than single developers when identifying code smells. The remainder of this section is organized as follows. Section 3.1.1 presents the study goal, the research question, and associated hypotheses. Section 3.1.2 discusses the characterization of subjects regarding topics which are relevant to our controlled experiment. Section 3.1.3 describes the target software systems and data sources used in the experiment. Section 3.1.4 presents the data analysis procedure. Finally, Section 3.1.5 describes the experiment procedure steps.

#### 3.1.1 Research Goal

The empirical study presented in this chapter aims at comparing both collaborative and individual smell identification. In summary, our goal was to understand whether collaborators are more effective than single developers when identifying code smells. By relying on the guidelines provided by Wohlin *et al.* (2012), we refined and structured the study goal as follows:

<sup>1</sup>The paper is conditioned to minor changes for publication.

- **Analyze** the collaborative smell identification when compared to individual smell identification,
- **For the purpose of** assessing the developers’ effectiveness,
- **With respect to** precision and recall of the identification of code smells,
- **From the viewpoint of** novice developers and professional developers,
- **In the context of** Java software systems that novice and professional developers are unfamiliar with, i.e., systems about which developers have no previous knowledge.

From our study goal, we designed the following research question (RQ).

**RQ.** Is collaborative smell identification more effective than individual smell identification?

Our empirical study addresses the RQ as follows. First, we assess the collaborative smell identification from the viewpoint of developers who are unfamiliar with the software systems under analysis. In other words, the developers have no previous knowledge about the software system in which they identify code smells. Second, we compute two metrics to compare developers effectiveness in code smell identification: precision and recall (23). Basically, precision measures the correctness of the identified code smells, and recall measures the completeness of the code smell identification with respect to all existing code smells based on the identification performed by an specialist in the software development or code smells. To compute these metrics, we built a reference list of code smells, i.e., an itemization of code smells thoughtfully identified in the software systems (25). Details on how we build the reference list can be found in Section 3.1.4. Third, we derived our null ( $H_0$ ) and alternative hypotheses ( $HA$ ) from  $RQ$  as presented in Table 3.1.

Table 3.1: Study hypotheses derived from  $RQ$

Hypothesis	Description
$H_0$	There is no difference in the effectiveness of collaborators or single developers in smell identification.
$HA_1$	There is a difference in the precision between collaborators and single developers in smell identification.
$HA_2$	There is a difference in the recall between collaborators and single developers in smell identification.

We discuss each hypothesis as follows. With the null hypothesis ( $H_0$ ), we assume that the number of developers working on the identification of code smells does not make it more or less effective. On the other hand, the alternative hypotheses indicate that there is a difference between the effectiveness of code smell identification by collaborators and single developers, with respect to precision ( $HA_1$ ) or recall ( $HA_2$ ).



### 3.1.2 Characterization of the Subjects

As aforementioned, this empirical study involved 26 developers as subjects. We classified the subjects according to two levels of working experience, namely *novice developers* and *professional developers*. This classification aimed at helping to understand whether each level can benefit differently (or not) from collaborative smell identification. It also aimed at supporting the generalization of our results, since we consider developers with different working experiences, i.e., novices to professionals. We introduce each level as follows.

- Novice developers are subjects with little or no experience in industrial software development. We selected these subjects from a software engineering course of a Brazilian undergraduate course in Computer Science.
- Professional developers are subjects currently acting in the industrial software development and that hold at least one year of experience – mostly due to the high developer turnover in the selected organization that made unfeasible selecting developers with a much more years of experience. We selected these subjects from a Brazilian software development organization with more than 10 years of activity, focused on maintaining data management systems, and with a development team that is truly concerned with identifying and eliminating code smells.

Our experiment consists of two sessions: one with novice developers in an academic laboratory, and another with professional developers in their working environment. For each session, subjects first performed smell identification in isolation and, after that, they performed the same task in collaboration. To participate in the study, all subjects signed an informal consent form (Appendix A). The subjects also filled out a characterization questionnaire with closed questions about their expertise in four topics related to the study: programming, Java, Pair Programming (PP), and code smells (Appendix B). We chose PP to introduce a concept of collaborative work, which is well known in both the literature and the industry (8, 15, 85).

Table 3.2 presents the data collected from the subject characterization questionnaire with respect to all subjects. The first column lists the two levels of working experience, i.e., novice developers (16 subjects in total) and professional developers (10 subjects in total). The second column provides a label for each subject, aimed at keeping anonymous their identity. The remaining columns present the experience reported by each subject regarding each aforementioned topic related to our study.

Table 3.2: Characterization of subjects

Working experience	Subjects	Topics			
		Programming	Java	Pair Programming	Code Smells
Novice developers	s1	Medium	Medium	Medium	Low
	s2	Medium	Medium	Medium	Medium
	s3	Low	Low	Medium	Low
	s4	Low	Low	Medium	Low
	s5	Low	Low	Medium	Low
	s6	Low	Low	Medium	Low
	s7	Low	Low	Medium	Low
	s8	Medium	Medium	Medium	Medium
	s9	Medium	Medium	Medium	Medium
	s10	Low	Low	Medium	Low
	s11	Medium	Medium	Medium	Medium
	s12	Medium	Medium	Medium	Low
	s13	Medium	Medium	Medium	Medium
	s14	Medium	Medium	Medium	Medium
	s15	Medium	Medium	Medium	Low
	s16	Medium	Medium	Medium	Medium
Professional developers	s17	High	High	Medium	High
	s18	High	High	Low	Medium
	s19	High	High	Low	Medium
	s20	High	High	Low	High
	s21	High	High	Medium	High
	s22	High	High	Low	High
	s23	High	High	Medium	Medium
	s24	High	High	High	High
	s25	High	High	High	High
	s26	High	High	Medium	High

We ranked the knowledge of subjects per topic (Table 3.2) in four categories, namely: *none*, *low*, *medium*, and *high*. Table 3.3 describes the categories, which we illustrate as follows. In the case of Java, the knowledge of the subject is: *none* when the subject never had contact with the Java programming language; *low* when the subject had contact with Java only through classes or by reading instructional material; *medium* when the subject had contact with Java only in the context of academic systems developed in academic courses and laboratories; and *high* when the subject had contact with Java for at least one year in industrial software systems. Note that professionals might not be experienced with Java in the industry but with another language. Similar reasoning applies to the other topics, such as code smells.

Table 3.3: Knowledge categorization about topics

Category	Description
None	I never had contact with it
Low	I had contact with it in classes or instructional material
Medium	I had contact with it in the context of academic system
High	I had contact with it for at least one year in industrial systems

We analyze Table 3.2 to identify subjects with high degree of knowledge about our topics of interest, collected via characterization questionnaire, when compared to other subjects. The table highlights these subjects in boldface. We say that a subject has a sufficient knowledge about the topics for study when the subject has a *medium* knowledge (Table 3.3) in at least two topics, since it represents at least a half of knowledge on the related topics. We observe that: 20 out of 26 subjects have medium to high knowledge in *programming* and *Java*; 22 out of 26 subjects have medium to high knowledge in *Pair Programming*; 16 out of 26 subjects have medium to high knowledge in *code smells*; and no subject has no knowledge in any of four topics. We then conclude that our subjects met the minimum requirements to take part in our experiment.

### 3.1.3

#### Target Software Systems and Data Sources

To allow us to investigate the collaborative smell identification, we have selected a set of target software systems and data sources for analysis. We present both the target software systems and data sources as follows.

**Target software system.** This study focuses on the identification of code smells by the subjects. Thus, we selected a set of target systems for usage by the subjects during such identification. For this purpose, we selected two industry systems, namely Java IO and Java Print, which belong to the Java Core project<sup>2</sup>. This selection relies on the following criteria. First, each system has to be open source, which allows the study replication. Second, each software system has to enable the identification of code smells using the Stench Blossom tool in its default settings (52), which relies the well-known detection strategies for code smells (37) and provides a visualization of the code smell suspects. Third, each software system has to be affected by multiple code smells.

To support the generality of our study findings, we selected software systems in which we identified different smell types, with varying granularity. We focus on the following smell types (29): *Data Clumps*, *Large Class*, *Long Method*, and *Message Chain*. Both *Large Class* and *Long Method* are intra-class smells, which locally affect a single class of the software system. In turn, *Data Clumps* and *Message Chain* are inter-class smells, which affect multiple classes. The selected smell types affect different code structures in a software system, such as methods and classes (29). All selected smell types are reportedly very frequent in software systems (91).

<sup>2</sup>In: <http://openjdk.java.net/groups/core-libs>

**Data Sources.** To conduct our data analysis, we collect experimental data of the subjects from different data sources, namely: the *subject characterization questionnaire*, the *code smells report questionnaire*, and the *follow-up questionnaire*. We combined the data obtained via these data sources to compensate their strengths and limitations. We describe each data source as follows.

- **Subject characterization questionnaire:** it is composed of questions aimed at characterizing each subject, in terms of their knowledge on topics of interesting, such as programming, Java, code smells, and Pair Programming (Appendix B).
- **Code smell report questionnaire:** it is a questionnaire aimed at collecting the list the code smells identified by the subjects during the code smell identification (Appendix E).
- **Follow-up questionnaire:**, it is composed of questions aimed at collecting the perception of subjects regarding the code smell identification conducted in the experiment (Appendix F).

#### 3.1.4 Data Analysis Procedures

As aforementioned, we conducted an empirical study which uses multiple data sources for data analysis. Thus, we carefully designed our data analysis procedures. We present each procedure as follows.

**Creation of a code smell reference list.** We built a code smell reference list to support the data analysis. For this propose, we recruited two researchers, which are PhD students with knowledge in software development and the identification of code smells. The researchers identified code smells in the selected projects in a complementary way. That is, one researcher conducted the manual smell identification, without tool support, and the other used the Stench Blossom tool (52). As a result, each researcher obtained a list of possible code smell suspects, which were not exactly the same due to the subjectiveness of the identification of code smells. To reach a consensus, we computed the agreement between the lists of code smell suspects reported by both researchers. For each smell suspect, we have an agreement whenever the developers have confirmed or refuted the suspect together. Conversely, we have a disagreement whenever the developers diverged in opinion without a consensus. After, the researchers conducted an open discussion to reach a consensus. Finally, we built the final code smell reference list.

**Quantitative Data Analysis.** Our study assesses the effectiveness of both collaborators and single developers on the identification of code smells. For this purpose, we compute the developers' effectiveness in terms of two well-known metrics, namely precision and recall (23). Precision measures the correctness of the identified code smells. Recall measures the completeness of the identified code smells with respect to all code smells which occur in a system (23). To compute these metrics, we used the aforementioned code smell reference list, which is an itemization of code smells identified in a systems (25).

Precision and recall were calculated based on the number of code smells marked as *true positive* (TP), *false positive* (FP) and *false negative* (FN). An *TP* occurs when the developer identifies a code smell that appears in the code smell reference list. An *FP* occurs when the developer identifies a code smell that does not appear in the code smell reference list. An *FN* occurs when a code smell appears in the code smell reference list but the developer was unable to identify. Both precision and recall are normalized in a range from 0 to 1. High precision values (close to 1) mean that the developer had reported, proportionally, only a few occurrences of FP in the software system. High recall values (close to 1) mean that the developer was able to identify a representative number of occurrences of TP in the software system. Equations 5-1 and 5-2 present the formula for precision and recall, respectively.

$$Precision = \frac{TP}{TP + FP} \quad (3-1)$$

$$Recall = \frac{TP}{TP + FN} \quad (3-2)$$

We applied the two-tailed Mann-Whitney test, which is a non-parametric statistical test, aimed at rejecting our null hypotheses ( $HA_0$ ) presented in Table 3.1. The reason for selected a non-parametric test is discussed as follows. Based on the normality test, we observed that both distributions of precision and recall are normal. We consider an alpha coefficient equal to 95%, which gives us a confidence interval of 5% (p-value < 0.05) to compare the data distributions. However, after applying the Levene's test (40), we observed that the distribution of recall is not homoscedastic, which requires the application of a non-parametric test. To avoid applying different statistical tests for precision and recall, and due to the limited sample of our study, we decided to apply the non-parametric test. We used the Minitab tool (38) to apply the statistical test.

**Complementary Data Analysis.** As aforementioned, we conducted a quantitative analysis on the effectiveness of both collaborative and the individual smell identification. In addition, we conducted a complementary analysis based on the follow-up questionnaire, which was applied after the experiment execution with the developers. This complementary analysis aimed at understanding the feedback of subjects regarding the experiment, mainly focused on the difficulties faced by the subjects to identify code smells. The analysis aimed at understanding the subject viewpoint on the identification of code smells, specially collaborative smell identification.

### 3.1.5 Experiment Steps

Figure 3.1 presents the four steps designed to guide our controlled experiment. As earlier discussed in this chapter, we conducted two sessions of the experiment, each with a group of participants classified according to their level of working experience (see Section 3.1.2 for details). The first session was conducted with the *novice developers*, while the second session was conducted with the *professional developers*. We asked all subjects to first fill out and sign a consent questionnaire. After that, the subjects engaged in the experiment. We describe in details each experimental step as follows.

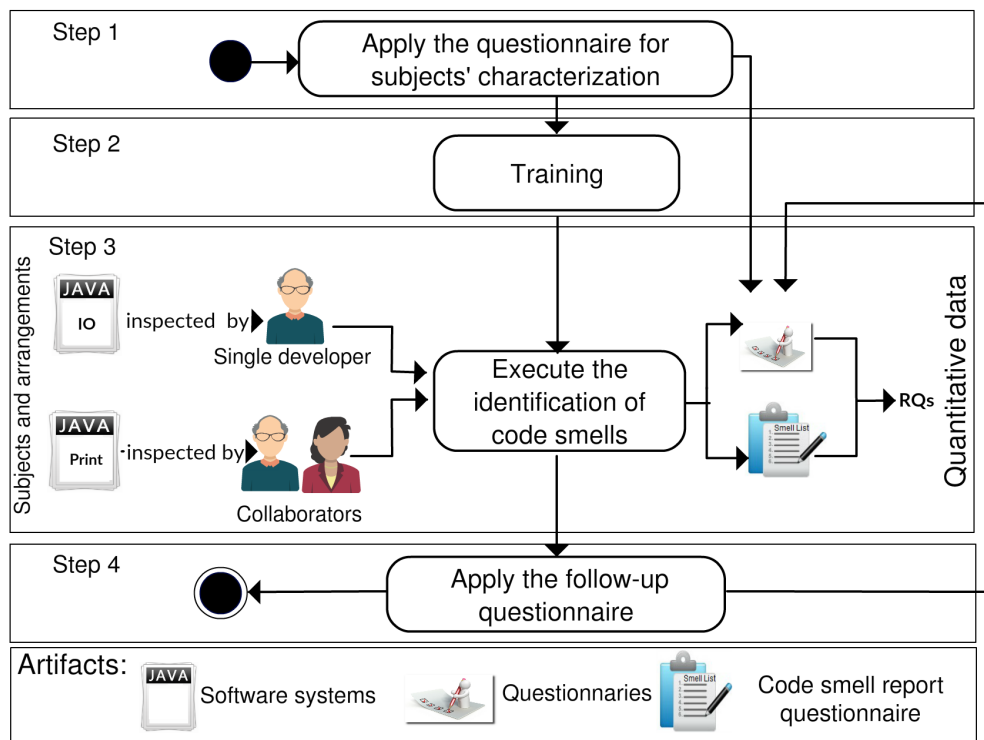


Figure 3.1: The experimental design

**Step 1. Apply the Subject Characterization Questionnaire.** The subject characterization questionnaire aims at characterizing each subject who participated in the experiment. The questionnaire includes questions regarding the background of the subject on programming, the Java programming language, pair programming, and code smells. The responses obtained through this questionnaire allowed us to identify some key characteristics of each subject, as presented in Section 3.1.2.

**Step 2. Training of Subjects.** After characterizing the subjects, we provided a training session to the subjects. This training aimed at supporting subjects to properly understand and execute the experiment. The training was organized in two parts. First, during 25 minutes, we explained the technical concepts and terminologies related to this study. Second, we took 10 minutes to conduct a discussion about the concepts. Overall, the training covered two topics, namely pair programming and code smells. Regarding code smells, we provided explicit definitions and practical examples (Appendix J). This training was provided to both novice and professional developers.

**Step 3. Smell identification task.** Per session of the experiment, we asked the subjects to engage in two experiment rounds of code smell identification. In the first round, all subjects individually inspected the Java IO project. In the second round, the same subjects collaboratively inspected the Java Print project. In each round, the subjects were asked to annotate the identified code smells in the code smell report questionnaire. This procedure allowed us to compute the number of true positives and false positives. We disposed the collaborators in the experiment similarly to a previous work (15), which paired the most experienced subjects with a less experienced one. All subjects performed the experiment simultaneously under the supervision of the researchers. Each round lasted 60 minutes only for the identification of code smell.

**Step 4. Answer the follow-up questionnaire.** After participating in the two rounds of the experiment, the participants received a follow-up questionnaire to fill. This questionnaire aimed at collecting the perception of each subject regarding the experiment. We aimed at understanding their opinion about the identification of code smells and the experience of working collaboratively to identify code smell. More details about this step are provided in (Appendix F).

## 3.2

### Effectiveness of Collaborative Smell Identification

This section presents the study results that answer *RQ*, which states: *Is the collaborative smell identification more effective than the individual smell identification?* Section 3.2.1 analyzes the distribution of precision and recall to confirm or refute the hypotheses of Table 3.1. Section 3.2.2 complements the findings based on the analysis per subject.

#### 3.2.1

##### Analysis of Distribution for Precision and Recall

At first, we analyzed the distribution of precision for collaborators and single developers. We aimed at understanding whether the collaborators tend to obtain a higher precision in the code smell identification when compared to single developers. We also computed the average precision for collaborators and single developers, by summing their precision regardless of their working experience and dividing this value by the total number of participants. Finally, we obtained the average precision for collaborators and single developers. Thus, we investigate the alternative hypothesis  $HA_1$  as follows.

**$HA_1$ . There is a difference in the precision between collaborators and single developers on the identification of code smells.**

Figure 3.2 presents the distribution of precision for collaborators and single developers, respectively. The figure also indicates the average precision for both collaborators and single developers. Overall, we observe an average precision equals 0.78 (78%) for collaborators against 0.59 (59%) for single developers. Our results suggest that collaborators had a 19.42% higher average precision than single developers. By applying the Mann-Whitney test, we observed a significant difference between precision values (p-value = 0.004). In summary, our results lead us to reject the null hypothesis  $H0_1$  and **accept the alternative hypothesis  $HA_1$** .

We analyze the distribution of recall for collaborators and single developers. Similarly to precision, we compute the average recall as follows. First, we sum the recall of developers regardless the working experience. Second, we divided this value by the total number of developers, which resulted in the average recall for both collaborators and single developers. Thus, we investigate the alternative hypothesis  $HA_2$  as follows.



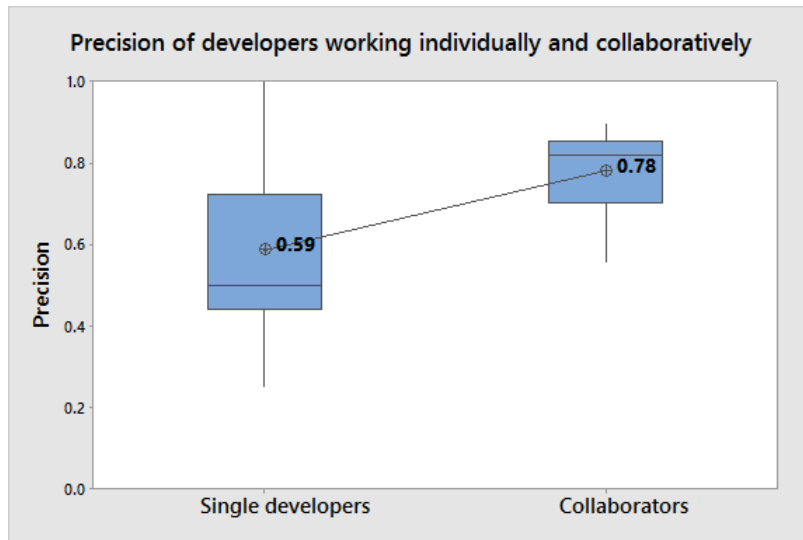


Figure 3.2: Distribution of precision for developers

**$HA_2$ . There is a difference in the recall between collaborators and single developers on the identification of code smells.**

Figure 3.3 presents the distribution of the average of recall. We observed an average recall equals 0.75 (75%) for collaborators against 0.27 (27%) for single developers. Our results suggest that collaborators had a 48.04% higher average recall than single developers. By applying the Mann-Whitney test, we observed significant difference between recall values ( $p$ -value = 0.001). Consequently, our results led us to reject the null hypothesis  $H_0$  and **accept the alternative hypothesis  $HA_2$** .

In summary, our results for precision and recall suggest that developers tend to identify more smells when working collaboratively. Particularly, we observed that collaborators obtained higher precision and recall than single developers. These results have two main implications discussed as follows. First, collaborators tend to make less mistakes when identifying code smells, i.e., they obtain higher precision. Second, collaborators are able to identify a more representative number of code smells in the software systems than single developers. In summary, our results lead us to Finding 1.

*Finding 1.* Collaborators tend to be more effective than single developers when identifying code smells in software systems.

Table 3.4 presents a complementary analysis per working experience (novice and professional developers), aimed at assessing any biases on the results of precision and recall caused by the working experience of the developers. The first column lists the working experience. The second column lists the experiment groups (individual and collaborators). The third and fourth columns

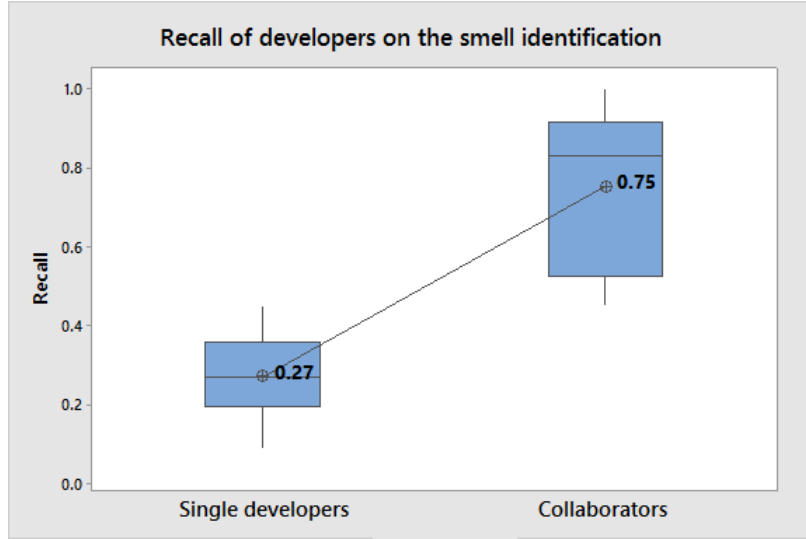


Figure 3.3: Distribution of recall for developers

present precision with respect to average and median precision. The fifth and sixth columns present recall with respect to average and median recall. We discuss our results as follows.

Working Experience	Group	Precision		Recall	
		Average	Median	Average	Median
Novice	Individual	0.61	0.50	0.23	0.27
	Collaborators	0.74	0.76	0.72	0.76
Professional	Individual	0.54	0.50	0.30	0.27
	Collaborators	0.85	0.83	0.80	0.90

Table 3.4: Comparison of precision and recall per working experience

In general, we observe that the results of the complementary analysis for both working experiences confirm Finding 1, i.e., they show that collaborators tend to have higher precision and recall than single developers. However, by comparing both working experiences, we observed a non-ignorable difference. In the case of single developers, there is difference in the results obtained with respect to the average precision and median recall that is equal to 7% in both cases. In the case of collaborators, there is difference in the results obtained with respect to both average and median precision (11% and 7%, respectively) and both average and median recall (14% and 8%). Thus, although working experience somehow affects Finding 1, it remains valid that collaboration improves precision and recall regardless the working experience.

### 3.2.2

#### Comparing Precision and Recall of Collaborators and Single Developers

After analyzing the distribution of precision and recall presented in Section 3.2.1, we conducted a more detailed analysis. We aimed at deeply understanding the effectiveness of the collaborative smell identification per developer, who may have performed the identification of code smells as collaborator and single developer. First, we analyze precision as follows.

Figure 3.4 presents the precision of collaborators and single developers per subject. For each set of three consecutive bars (two black and one gray bar), we compare the precision of two developers as single developers with the precision of both developers as collaborators. Overall, 8 of the 13 sets (61.53%) obtained higher precision as collaborators. In addition, for 4 of the 5 remaining sets (38.47% of the total), at least one developer as single developer have improved its precision when worked as a collaborator (by comparing the gray bar of the single developers with the black bar of the corresponding collaboration). It implies that collaboration improves the effectiveness of at least one developer involved in the collaboration (as observed for 4 out of the 5 cases), by reducing the number of incorrectly identified code smells.

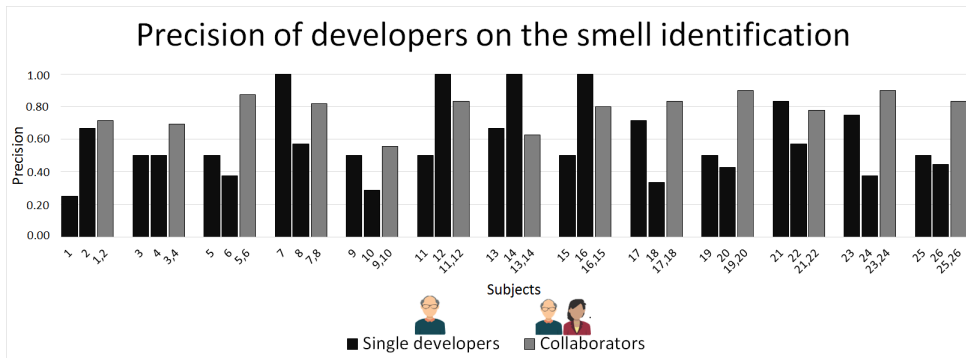


Figure 3.4: Precision of collaborators and single developers per subject

By analyzing the follow-up questionnaire, we may draw additional conclusion on the benefits of the collaborative smell identification. All subjects stated that the collaboration minimized their frustrations and improved their confidence during the identification of code smells. For example, subject s16 said: *Pair programming has strengthened the communication between members and the possibility of a more precise analysis because four eyes see more than two... consequently we were more confident in our work.* In turn, subject s20 said: *The discussions with my partner were essential for understanding the long chaining of methods and for confirming the existence of Message Chain.*

Next, we analyzed recall as follows. Figure 3.5 presents the recall values obtained by collaborators and single developers. Each set of three consecutive

bars (two gray and one black bar) compares the recall of two subjects working as single developers with the recall of both together working as collaborators. Overall, 100% of the sets obtained higher recall for collaborators than single developers. It reinforces our findings of Figure 3.3 and suggests that collaborators tend to identify a more representative number of code smells in the software systems, when compared with single developers.

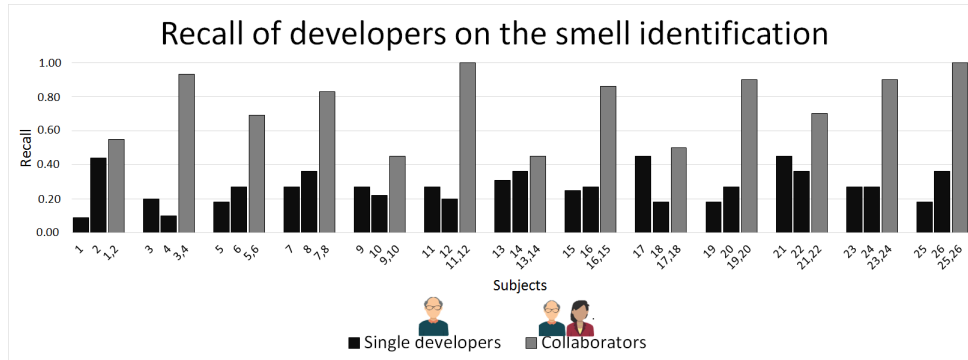


Figure 3.5: Recall of collaborators and single developers per subject

By analyzing the follow-up questionnaire, we draw the following observations. We found that, during the collaborative smell identification, one collaborator was usually responsible for selecting a code smell suspect and, after, both collaborators started arguing about the code smell suspect. We have also found that collaborators have more confidence to confirm a code smell suspect when compared with single developers. Consequently, collaborators are able to identify a larger number of code smells in software systems. This observation is reinforced by the opinion of the subjects, such as s7 that said: *The greatest potential of working collaboratively was the possibility of adding different strategies to determine a code smell. This fact was only possible thanks to different experiences that each one of us has.*

Finally, the follow-up form revealed the certainties and uncertainties of the subjects about each code smell suspect. For instance, when the collaborators were uncertain on confirming a code smell suspect, both ended up not confirming a particular code smell suspect as an actual code smell. Thus, by relying on the comments of subjects like s7 and s20, we conclude that collaborators may exchange information and, consequently, improve their effectiveness when compared with single developers, which leads us to Finding 2.

*Finding 2.* The exchange of information among collaborators has a potential to improve the effectiveness of the code smell identification.

### 3.3

#### Threats to Validity

Even with a careful planning, any empirical study is affected by threats which may invalidate the study findings. We discuss threats to the validity of our study and the respective minimizations as follows (89).

**Construct Validity.** We have restricted our study to the analysis of a limited set of code smell types, which may have affected our findings. However, we minimize this threat by selecting four well-known and varied code smell types, such as *Large Class* and *Message Chain*. These types occur in different code elements and are reportedly common in software systems (Section 3.1.3). Regarding the creation of the code smell reference list, we recruited two PhD students with knowledge in software development and the identification of code smells. Thus, we mitigate possible threats by engaging researchers that are sufficiently qualified for such creation (Section 3.1.4). Finally, with respect to the different background of the subjects, we mitigate this threat by selecting subjects with at least a minimum knowledge on topics of interest, such as Java and code smells (Section 3.1.2). In addition, all subjects underwent the training sessions to normalize their background.

**Internal Validity.** Regarding the communication among subjects during the experiment execution, we mitigate threats by limiting such communication with little interference on their answers. We also explained the experimental tasks for all subjects, aimed at avoiding misunderstandings, and reduced the communication among subjects. With respect to the experiment execution, we designed experimental tasks that fit our time constraints of one hour, by relying on experiment simulations performed with volunteers. Through these simulations, we also identified opportunities for improving the experiment with our final subjects. Finally, regarding the developer arrangement, we did not apply a cross-over design (93) in our study. However, to minimize learning biases of developers identifying code smells individually and in pairs, we selected different systems for analysis.

**Conclusion Validity.** To conduct the data analysis, we carefully selected the most appropriate statistical tests. We also paid special attention to avoid violating assumptions of the selected statistical tests. To answer our research question, we applied the Mann-Whitney test (76) as discussed in Section 3.1.4. Furthermore, we believe that our questionnaires fit our expectations with the empirical study and support answering our research question. For instance,

they allowed us to characterize the experienced and inexperienced developers. Thus, we mitigate possible threats related to the data analysis through the exclusive analysis of data collected from the questionnaires.

**External Validity.** With respect to the generalization of our study findings, the empirical study has some threats, as follows. First, we applied the study only in the context of Brazilian developers, which may not represent any development scenarios around the world due to cultural variations. In addition, although we have spent a period of one month to engage novice developers and professional developers in our study, the set of subjects is limited to 16 novice developers and 10 professional developers. We minimize possible threats regarding the set of subjects by involving developers with varied background and level of working experience. We also focused on developers with minimum experience with topics of interest, such as code smells and pair programming.

### 3.4 Summary

This chapter investigated whether collaborators are more effective than single developers on the identification of code smells. For this purpose, we present a controlled experiment conducted with 26 developers, which are categorized as novice and professional and developers, aimed at understanding to what extent the collaboration improves the developers' effectiveness when identifying code smells in software systems they are unfamiliar with. We compared the effectiveness of collaborators and single developers to answer the first research question of this thesis, namely: *Does collaborative smell identification improve developers' effectiveness when compared to individual smell identification?* We compute effectiveness in terms of precision and recall.

Our results suggest that collaborators tend to be more effective than single developers on the identification of code smells. In fact, we observed that collaborators may correctly identify more code smells, with a lower number of misidentified code smells, when compared with single developers. We summarize our study findings as follows.

- The average precision of collaborators was 19.42% higher than the average of single developers on the identification of code smells. This result has statistical significance, when considering a 95% confidence interval, i.e.,  $p\text{-value} < 0.05$ . In other words, collaborators tend to identify more actual code smells than single developers.
- The average recall of collaborators was 48.04% higher than the average of single developers on the identification of code smells. This result has

statistical significance (p-value  $< 0.05$  for a 95% confidence interval). That is, the identification of code smells performed by collaborators has a higher coverage than by single developers.

- The exchange of information allowed by collaboration is essential to improve the effectiveness of the code smell identification. We observed that collaborators share knowledge and complement each other. Consequently, it improves their confidence on confirming a code smell suspect.

All aforementioned study findings raise relevant questions. For instance, what factors help to improve the effectiveness of collaborative smell identification? These factors might be the granularity of code smells, the developers team size, or the familiarity of developers with the software systems under analysis. Since collaborators are more effective than single developers on the identification of code smells, there is a need for understanding what circumstances lead to the improvement of effectiveness when developers work collaboratively. We address this need in the next chapter.

## 4

# Understanding the Effectiveness of Collaborative Smell Identification

Software systems might become difficult to maintain due to the occurrence of code smells (44), which should be identified and eliminated whenever possible (29). However, identifying code smells is often difficult for a single developer (17, 63), since it requires analyzing several code elements developed by different developers. Thus, the effectiveness of the identification of code smells could benefit from the knowledge exchange among collaborators (Section 1.1). In Chapter 3, we provided empirical evidence on the effectiveness of collaborative smell identification. In fact, we observed that collaborators tend to obtain higher precision and recall than single developers on the identification of code smells, regardless of their working experience.

After identifying evidence mostly in a quantitative way, we adapted the design of our previous study (presented in Chapter 3) to enable us to better understand the effectiveness of collaborative smell identification in a quantitative and qualitatively way. Indeed, we investigated the developers' effectiveness with the aim of revealing some influential factors and collaborative activities (Section 2.1) performed by developers when identifying smells together.

As a first step in this chapter, we answer the second specific research question, which states: ***SRQ<sub>2</sub>*: What influential factors contribute to the developers' effectiveness in the collaborative smell identification?** For this purpose, we conduct an empirical study with novice developers, who had to identify code smells in software systems they are unfamiliar with. With respect to the influential factors, we are concerned with three factors which capture different characteristics of software developers and identification of code smells: (i) the characteristics of the software systems, (ii) the developers team size, and (iii) the smell granularity. We will investigate whether these factors may cause any effect on collaborative smell identification. Our goal is to be able to identify and summarize these influential factors, which may contribute even more to the effectiveness of collaborative smell identification.

After identifying the influential factors which contribute to the collaborative smell identification, we answer the third specific research question of this thesis, which states: ***SRQ<sub>3</sub>*: What collaborative activities contribute**



**to the developers' effectiveness on smell identification?** By answering this question, we will be able to characterize the smell identification activities recurrently performed by either single developers or collaborators. In fact, one of our goals is to identify collaborative activities related with those influential factors, which may improve the effectiveness of smell identification. Moreover, we aimed at understanding to what extent the activities currently performed by developers in isolation could benefit from their collaboration.

This chapter reports our second controlled experiment designed to help us understand the benefits and drawbacks of collaborative smell identification in more depth. The findings of this study can be summarized as follows.

- Collaborators achieved more effectiveness on the identification of inter-class smells (Section 2.1). Indeed, collaborators identify a higher average number of code smells because they almost always had to consider both of each developer's knowledge on revealing scattered, complementary symptoms associated with a single smell type.
- We also derive empirical evidence that adding more than two developers on the task of collaborative smell identification does not necessarily improve their identification effectiveness.
- Developers perform several collaborative activities during the identification of code smells. However, for some activities, developers have limited support to conduct the collaborative smell identification.

The empirical study reported in this chapter was published in the 10th edition of the Brazilian Symposium on Components, Architectures, and Reuse (SBCARS) (62). The remainder of this section is organized as follows. Section 4.1 describes the study settings, including the study goal, research questions, and study steps. Section 4.2 presents the results of our empirical study with respect to the effectiveness of the collaborative smell identification. Section 4.3 discusses threats to the study validity. Finally, Section 4.4 summarizes this chapter and introduces the following chapter.

## 4.1 Study Settings

This section describes the settings of our empirical study aimed at characterizing the influential factors and the collaborative activities, which improve the effectiveness on the identification of code smells. Section 4.1.1 presents the study goal, research questions, and associated hypotheses. Section 4.1.2 discusses the characterization of subjects regarding topics which are relevant to our controlled experiment. Section 4.1.3 describes the target software systems

and data sources used in this experiment. Section 4.1.4 presents the procedures of data analysis. Finally, Section 4.1.5 describes the experiment steps.

#### 4.1.1 Research Goals

This chapter presents an empirical study aimed at characterizing the developers' effectiveness, focus on the influential factors and collaborative activities of the identification of code smells. Based on the guidelines provided by Wohlin *et al.* (2012), we designed the following study goal:

- **Analyze** collaborative smell identification when compared to individual,
- **For the purpose of** characterizing the influential factors on the effectiveness of the identification of code smells and the collaborative activities that contribute to such identification,
- **With respect to** the average number of identified code smells,
- **From the viewpoint of** novice developers,
- **In the context of** Java systems that novice developers are unfamiliar with, i.e., those systems which developers lack previous knowledge about.

From our study goal, we designed the following research questions ( $RQ_1$ ).

**$RQ_1$ .** Do collaborators identify more code smells than single developers?

We answer  $RQ_1$  by comparing the results of single developers and collaborators based on the influential factors presented in the introduction of this chapter. This first question is somewhat similar to the research question addressed in Chapter 3, which enables us to understand whether the results in both experiments were similar or not. We assess collaborative smell identification focused on developers that are unfamiliar with the analyzed systems. After, we computed the average number of identified code smells to compare the effectiveness of collaborators and single developers. We compute the average number of identified code smells by building a reference list of code smells (Section 2.1). Section 4.1.4 details how we build the reference list in this particular experiment. Finally, we derived our null ( $H_0$ ) and alternative hypotheses ( $HA$ ) from  $RQ_1$  as presented in Table 4.1.

In a way, the previous research question is similar to the one addressed in Chapter 3. However, we need to get in-depth knowledge about phenomena associated with collaborative smell identification. Previous work discussed in Chapter 2.1 characterizes the identification of code smells into three steps (65). However, there is little empirical evidence on how developers actually perform

Table 4.1: Hypotheses of  $RQ_1$ 

Hypothesis	Description
$H_0$	There is no difference in the average number of code smells identified by single developers and collaborators.
$HA_1$	There is a difference in the average number of code smell identified by single developers and collaborators.

the identification of a code smell. More importantly, researchers and practitioners do not have systematic knowledge about the main activities performed by both collaborators and single developers. Thus, with  $RQ_2$ , we aimed at revealing the collaborative activities typically involved in the identification of code smells. Our goal is to provide organizations with a deeper understanding about how to conduct collaborative smell identification in an effective way. This empirical study addresses the specific question  $SRQ_3$  of this thesis.

**RQ<sub>2</sub>.** How do single developers and collaborators perform the identification of code smells?

#### 4.1.2

##### Characterization of the Subjects

Our empirical study involved 28 novice developers as subjects. We selected novice developers due to the following reasons. First, software developers begin a transition from novice to professional at least twice in their careers: in their first year in the university, and when they are about to start their first industrial system (9). Thus, experienced developers are never available in the former, and they might not be available in training courses in the second transition. An example of this second transition are software projects run by companies like ThoughtWorks/Kaizen (21), which provided the context of our present experiment. Second, in our previous study reported in Chapter 3, we have found that collaboration improves the developer effectiveness on the identification of code smells regardless the working experience. Thus, we decided to conduct this study with novice developers only categorized as follows.

- Novice developers with some experience in the development of industrial software systems. We selected these subjects from a software development course of a Brazilian university (21). This course provides undergraduate students in Computer Science (CS) an immersion that lasts four months in the development of industrial software systems in the context of ThoughtWorks/Kaizen software projects.
- Novice developers without experience in the development of industrial software systems. We selected these subjects from a software engineering course of a Brazilian undergraduate course in CS.

Our experiment consisted of two sessions, each with a different set of subjects. The first session was conducted with novice developers with experience in the development of industrial software systems. The second session was conducted with novice developers without experience in the development of industrial software systems in an academic laboratory. To participate in the study, all subjects signed a consent questionnaire (Appendix A). The subjects also filled out a characterization questionnaire with closed questions about their experience in three topics related to the study: programming, Java, and Pair Programming (PP) (Appendix C).

Table 4.2 presents the data collected from the subject characterization questionnaire, with respect to all subjects. The first column lists the two levels of working experience. The second column provides a label for each subjects, aimed at keeping anonymous the identify of subjects. The remaining columns presents the experience reported by each subjects regarding each aforementioned topic related to our study. We ranked the knowledge of subjects per topic of Table 4.2 in four categories: *none*, *low*, *medium*, and *high*. We explain in detail each category as follows.

Table 4.3 describes the categories, which we illustrate as follows. For instance, in the case of programming, the knowledge of the subject is: *none* when the subject never had contact with programming; *low* when the subject had contact with programming only through classes or by reading instructional material; *medium* when the subject had contact with programming only in the context of academic systems developed in academic courses and laboratories; and *high* when the subject had contact with programming in the context of industrial software systems. Similar reasoning applies to the other topics related with this study, such as Java. Additionally to Table 4.3, we observed that all subjects already have a minimum knowledge on code smells from classes. Thus, we do not show this data in the table.

Back to Table 4.2, we identify the subjects with high knowledge about our topics of interest, collected via characterization questionnaire, when compared to other subjects. The table highlights these subjects in boldface font. We say that a subject has the highest knowledge of the topics when the subject has a *medium* knowledge (Table 4.3) in at least two of the three topics. This analysis lead us to the following observations. First, 22 out of 28 subjects have medium to high knowledge in *programming* and *Java*. Second, 23 out of 28 subjects have medium to high knowledge in *Pair Programming*. Third, no subject has no knowledge in any of three topics. We conclude that our subjects met the minimum requirements to participate in the experiment.

Table 4.2: Characterization of subjects

Working Experience	Subjects	Topics		
		Programming	Java	Pair programming
Novice developers with experience in industrial software	s1	High	High	Medium
	s2	High	High	Medium
	s3	High	High	Medium
	s4	High	High	Low
	s5	High	High	Low
	s6	High	High	Medium
	s7	High	High	Low
	s8	High	High	Medium
	s9	High	High	Low
	s10	High	High	Medium
	s11	High	High	Medium
	s12	High	High	Low
	s13	High	High	Low
	s14	High	High	Medium
Novice developers without experience in industrial software	s15	Low	Low	Medium
	s16	Medium	Medium	Low
	s17	Medium	Medium	Medium
	s18	Low	Low	Medium
	s19	Medium	Low	Low
	s20	Low	Low	Medium
	s21	Medium	Medium	Low
	s22	Low	Low	Low
	s23	Medium	Medium	Low
	s24	Medium	Medium	Low
	s25	Medium	Medium	Low
	s26	Low	Low	Medium
	s27	Medium	Medium	Low
	s28	Medium	Medium	Low

### 4.1.3

#### Target Software Systems and Data Sources

In order to allow us to investigate collaborative smell identification, we defined the target software systems and data sources as follows.

**Target Software Systems.** For study purposes, we developed three software systems to be used by the subjects during the identification of code smells. The systems are named A, B, and C. Table 4.4 provides general data about each software system. The first column lists the software systems. The second column describes their purpose. The third and forth columns represent the number of methods and classes per system. For instance, system C aims at managing bookstore stocks, and contains 8 classes and 60 methods. It is important to highlight that we could not choose more complex systems give the typical time constraints of a controlled experiment.

These systems are inspired in requirements provided by the literature (20), address different domains and were implemented using the Model-View-

Table 4.3: Knowledge categorization about Topics

Category	Description
None	I never had contact with it
Low	I had contact with it in classes or instructional material
Medium	I had contact with it in the context of academic system
High	I had contact with it in the context of industrial software systems

Table 4.4: General data about each software system

System	Purpose	Methods	Classes
A	Educational guessing game	38	7
B	Bank automatic teller machine simulator	32	6
C	Stock management system of a bookstore	60	8

Controller architectural pattern (35). These systems were developed based on the following criteria. First, the software systems were developed in the Java programming language. Second, each software system has to enable the identification of code smells using the Stench Blossom tool in its default settings (52), which reports and provides a visualization of the code smell suspects. Third, each software system has to be affected by multiple code smells. Fourth, the systems should have a sufficient size to support subjects in understanding the behavior of the system and conduct the empirical study within time constraints. Previous work states that even small-sized systems may have multiple code smells, and their occurrence patterns do not differ from what is observed in large-sized systems (44).

To support the generality of our study findings, we developed systems affected by different smell types, with varying granularity. We focused on smell types defined by Fowler (29). *Long Method* and *Duplicated Code*, which are intra-class smells, which locally affect a single class of the system. Depending on the context, *Duplicated Code* can be classified as intra-class or inter-class (29). We have considered it as inter-class cases did not naturally occur in the analyzed projects in this experiment. That is, we did not take into account code duplication that occurs between two different classes. In turn, *Data Class*, *Feature Envy*, *Lazy Class*, and *Intensive Coupling* are inter-class smells, which affect multiple classes. The selected smell types affect different code structures in a system, such as methods and classes (29). In addition, all selected smell types are reportedly very frequent in software systems (91).

**Data Sources.** In order to conduct our data analysis, we collected experimental data from different data sources, namely: the *subject characterization questionnaire*, the *code smells report questionnaire*, the *follow-up questionnaire*, and multimedia data such as *audio* and *videos* of the experiment sessions.

We combined the data obtained via these data sources to compensate their strengths and limitations. For instance, a questionnaire mostly provides quantitative data, but a video can add qualitative data about the identification of code smells. We describe each data source as follows:

- **Subject characterization questionnaire:** it aimed at characterizing each subject, in terms of their knowledge on topics of interest, such as programming and Java (Appendix C).
- **Code smell report questionnaire:** it is a questionnaire aimed at collecting the list of code smells identified by the subjects during the experimental tasks (Appendix E).
- **Follow-up questionnaire:** it is composed of questions aimed at collecting the impression of subjects regarding the code smell identification conducted in the experiment (Appendix G).
- **Audio and Video:** it includes screenshots provided by Camtasia<sup>3</sup> and audio and video records regarding the actions of each subject.

#### 4.1.4 Data Analysis Procedures

In order to compute the quantitative data of our study, we designed the data analysis procedures presented as follows.

**Creation of a code smell reference list.** We created a code smell reference list per system. We recruited two researchers with experience in the identification of code smells to run the three following steps. First, the researchers used a smell detection tool (52) to identify code smell suspects. Second, they confirmed or refuted, in isolation, each code smell suspect. Each researcher then obtained a list of code smell suspects, which could differ since the identification of code smells is subjective. Third, we computed the agreement between the lists obtained by the researchers. Both researchers discussed how to solve any conflicts and to reach a consensus. There was no case where consensus was not achieved. Finally, we created the code smell reference list.

**Quantitative Data Analysis.** To identify the influential factors on the collaborative smell identification and the collaborative activities, we assessed the average number of identified code smells per smell type. We computed the average total number of identified code smells divided by the total number

<sup>3</sup>In. <https://www.techsmith.com/camtasia.html>

of subjects involved in the identification of code smells. Differently from our previous study reported in Chapter 3, which concerns the identification of code smells per subject, we conduct an overall analysis of the average number of identified code smells to investigate the hypotheses of Section 4.1.1.

To test our hypotheses, we conducted the statistical analysis with the support of Minitab (38). We decided to apply non-parametric statistical tests once all data follow a normal distribution, but were not homoscedastic. Thus, we applied the two-tailed Mann-Whitney ( $\alpha = 95\%$ ) to compare the data distributions. We also applied the Kruskal Wallis (K-W) statistic test specifically for  $HA_1$ , aimed at assessing the number of code smells identified in multiple software systems (Systems A, B, and C) and considering three subject arrangements, which are single developers, pairs, and groups (Section 4.1.5).

**Qualitative Data Analysis.** Our qualitative data analysis relies on the following artifacts. First, the characterization questionnaire, aimed at collecting the participant background. Second, the follow-up questionnaire, aimed at collecting the perception of developers regarding the identification of code smells and the collaborative smell identification. Third, screen shots, and multimedia data such as audio and video records, which were collected during the identification of code smells for both single developers and collaborators. These last data sources were used to support understanding how developers conduct the identification of code smells and help comprehend the quantitative data.

All data were transcribed and validated by researchers to eliminate problems in the data transcription. After that, we apply a technique for categorizing the transcribed data (71). This technique is called by the authors as a rule guided qualitative text analysis, which consists of, from a research question (RQ), determining what data categories are relevant for answering the RQ. In the context of our study, the categories could be the collaborative activities, for instance. After, analyzing the data collected from different data sources to categorize all data based and interpret the study results, which includes understanding how developers identify code smells and what activities they perform as single developers or collaborators. Finally, we obtained the set of collaborative activities of smell identification.

#### 4.1.5 Experiment Steps

We conducted two sessions of the experiment with different subjects. The first one involved novice developers without experience in the development of industrial software systems, whereas the second one involved novice developers



who worked in the development of at least one industrial software system. As discussed in Section 2.1, we characterized the collaborative smell identification in two ways: pairs, which consists of exactly two developers together, and groups, which comprise more than two developers working together to identify code smells. We aim to understand whether adding more than two developers improves the effectiveness of code smell identification.

Before participating in our study, the subjects were asked to fill out and sign a consent questionnaire (Appendix A). After, the subjects engaged in the experiment. Figure 4.1 presents the four steps designed to guide our controlled experiment. We describe in detail each experimental step as follows:

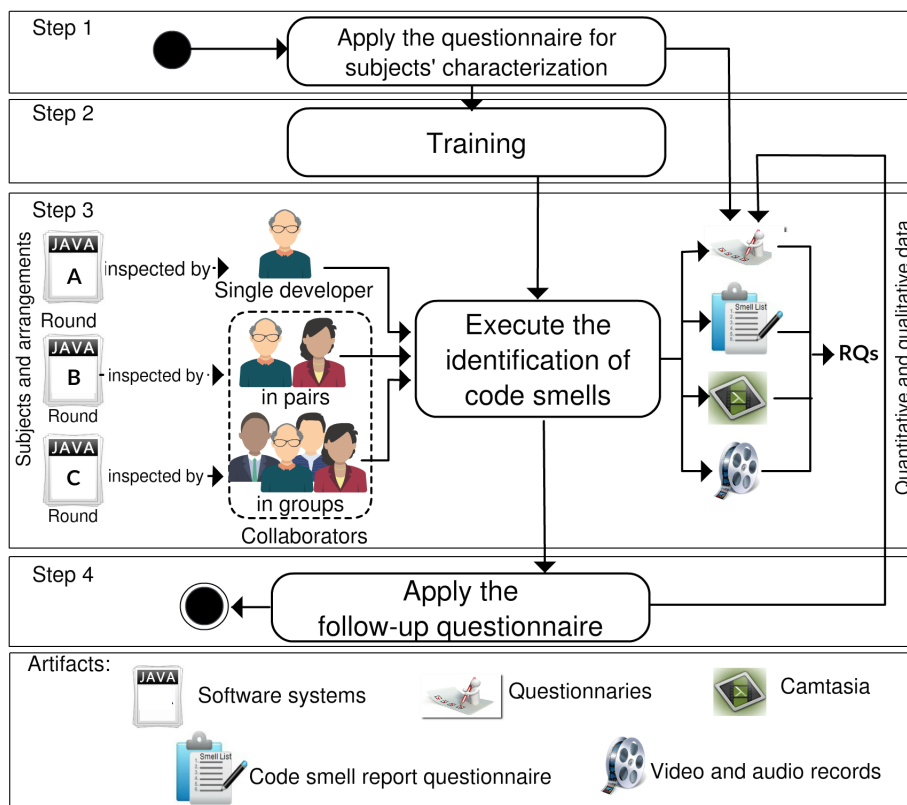


Figure 4.1: Study steps

- **Step 1** consists of applying the characterization questionnaire, which contains questions regarding the subject background on programming, the Java language, and pair programming.
- **Step 2** consists of training the subjects to participate in the experiment. This step aimed at equalizing the background of all subjects with respect to topics of interest of our empirical study.
- **Step 3** consists of identifying code smells performed by collaborators and single developers. All subjects identified code smells in three rounds,

which vary according to the three subject arrangements: single developer, pair, and group. Figure 4.1 highlights in a dotted box the two subject arrangements for collaborators: pairs and groups.

- **Step 4** consists of applying the follow-up questionnaire, which enables us to understand the subjects' viewpoints about the identification of code smells. Appendix G provides additional detail on the questionnaire.

With respect to Step 2, all subjects received training about the basic and empirical study concepts which are relevant in the context of this thesis. The subject training was organized in two parts. In the first part, we introduced relevant concepts, with focusing on pair programming and code smells (see Appendix J). With respect to pair programming, we discussed how developers have worked together to develop and review the same code element. In the case of code smells, we introduced the definitions extracted from the literature and presented practical examples. The first part lasted 25 minutes. In the second part, we promoted the discussion among all subjects about the introduced concepts. The second part of the training lasted 10 minutes.

Regarding Step 3, Table 4.5 presents the subject arrangement applied to our study subjects. This table is split into two parts: session 1, which was performed by novice developers with experience in industrial software systems; and session 2, which was performed by novice developers without experience in industrial software systems. Both parts of the table share the same columns, described as follows. The first column differentiates single developers and collaborators. The second, third, and fourth columns present the subjects arrangements for rounds 1, round 2, and round 3, respectively. We discuss how the subjects were arranged in each round as follows.

About the subject arrangement, each subject had to work in all three aforementioned rounds on the identification of code smells. This procedure aimed at minimizing threats to validity concerning the distribution of subjects among rounds and arrangements, in addition to the limited number of available subjects. Per round, the subjects were asked to annotate the identified code smells in the code smell report questionnaire. Our study was conducted on-line, i.e. simultaneously, and was completely supervised by the researchers. At the end of the experiment, the code smell report questionnaire was delivered to the researchers. In total, the identification of code smells lasted 60 minutes.

Back to Table 4.5, we illustrate how the subjects were arranged with an example as follows. Let us consider session 1, which was performed by novice developers with experience in industrial software systems. In round 1, the subjects s1 to s6 inspected system A as single developers. Later, in round 1, the same subjects inspected system B as collaborators, specifically

Table 4.5: Subject arrangement

<b>Session 1. Novice developers with experience in industrial software systems</b>			
<b>Subject arrangement</b>	<b>System A - round 1 -</b>	<b>System B - round 2 -</b>	<b>System C - round 3 -</b>
Single developers	Subjects s1, s2, s3, s4, s5, s6	Subjects s7, s8, s9, s10	Subjects s11, s12, s13, s14
Collaborators	s11 and s12 s13 and s14	s1 and s2 s3 and s4 s5 and s6	s7 and s8 s9 and s10
	s7, s8, s9 and s10	s11, s12, s13 and s14	s1, s2, s3, s4, s5 and s6
<b>Session 2. Novice developers without experience in industrial software systems</b>			
<b>Subject arrangement</b>	<b>System A - round 1 -</b>	<b>System B - round 2 -</b>	<b>System C - round 3 -</b>
Single developers	Subjects s15, s16, s17, s18, s19, s20	Subjects s21, s22, s23, s24	Subjects s25, s26, s27, s28
Collaborators	s25 and s26 s27 and s28	s15 and s16 s17 and s18 s19 and s19	s21 and s22 s23 and s24
	s21, s22, s23 and s24	s25, s26, s27 and s28	s15, s16, s17, s18, s19 and 20

allocated in pairs. Finally, in round 3, the same subjects inspected system C as collaborators, but now structured as a group. The other subjects were similarly arranged among rounds of the study.

## 4.2

### Influential Factors and Collaborative Activities

This chapter complements the study findings of Chapter 3 by summarizing the empirical evidence collected about the influential factors and collaborative activities. Section 4.2.1 presents the influential factors on the collaborative smell identification. Section 4.2.2 presents the collaborative activities that seem to improve the effectiveness of collaborative smell identification.

#### 4.2.1

##### Influential Factors on the Identification of Code Smells

This section presents and discusses the main results of our empirical study aimed at identifying the influential factors in the identification of code smells. Consequently, we address  $RQ_1$  as follows.

**RQ<sub>1</sub>.** *Do collaborators identify more code smells than single developers?*

**Average Number of Identified Code Smells as Indicators of Influential Factors.** Table 4.6 presents the average number of code smells identified by the subjects. The first column lists the three subject arrangements, namely individual, pairs, and groups. The other columns present the average number of code smells identified in each software system, namely systems A, B, and C. Regarding the overall number of code smells identified per arrangement, we have an interesting observation. The Kruskal-Wallis statistical test (see Section 4.1.4 for details) suggests differences in the number of identified code smells among different arrangements, which applies to all systems. In fact, we obtained p-values equals to 0.0006, 0.0064, and 0.0009 for systems A, B and C, respectively. This observation leads us to conduct additional tests aimed at understanding whether such differences address our alternative hypothesis.

Table 4.6: Average number of identified code smells

Subject Arrangement	System A	System B	System C
Single developers	3.33	1.28	6.88
Collaborators in pairs	7.25	5.00	14.00
Collaborators in groups	6.00	3.00	5.00

*Influential Factor 1.* Similarly to our previous findings, we observe that collaborators are more effective than single developers in identifying code smells.

Data presented in Table 4.6 suggest that collaborators identified a higher average number of code smells than single developers. By comparing single developers with collaborators in pairs, we observe that collaborators identified more than twice as many code smells than single developers. In fact, collaborators identified up to four times more code smells than single developers in the case of system B. In addition, by comparing single developers with collaborators in groups, we observe that collaborators had an average number of identified code smells at least twice as large than the average number for single developers, in the case of systems A and B.

Overall, we observe that the average number of code smells identified by collaborators was higher when compared to single developers, except for system C, which coincidentally had more collaborators than systems A and B. Hence, our data suggest the adding more than two developers to the collaborative smells identification does not necessarily improve the effectiveness of such identification. In fact, our qualitative analysis indicated that the convergence among collaborators is often time consuming and the discussions often become

unproductive. It seems that there is often a pair of developers who are the most appropriate peers to perform the smell identification tasks. However, particularities of system C may have affected the results, which reinforce the value of our subject arrangement with different rounds. In summary, our data lead us to reject  $H_0$  and **accept the alternative hypothesis  $HA_1$** .

*Influential Factor 2.* Adding more than two developers to a team do not necessarily improve the effectiveness of smell identification.

With respect to collaborators working in pairs, we have found a significantly higher average number of smells identified for all systems, with p-values equal to 0.0001, 0.0015, and 0.0059, respectively. Consequently, we confirm our hypothesis with a confidence level of 95% for all systems. In turn, regarding the collaborators working in groups, we have found a significantly higher average number of code smells for two out of the three systems, with p-values equal 0.0004, 0.0029, and 0.8520, respectively. Again, we confirm our hypothesis with a confidence level of 95%, but only for systems A and B.

**Additional Influential Factors.** As aforementioned, we identified two influential factors by analyzing the average number of identified code smells, regardless of the smell type. Complementary, we investigate additional influential factors by analyzing the average number of identified code smells per smell type. Figure 4.2 presents the average number of identified code smells for all systems, within the time constraints of the experiment. The blue bars represent the results for single developers. The orange bars represent the results for collaborators working in pairs. The gray bars represent the results for collaborators working in groups. This figure represents only the results when the average number of identified code smells was equal or higher than 1.0 for at least one subject arrangement. We discuss our main findings as follows.

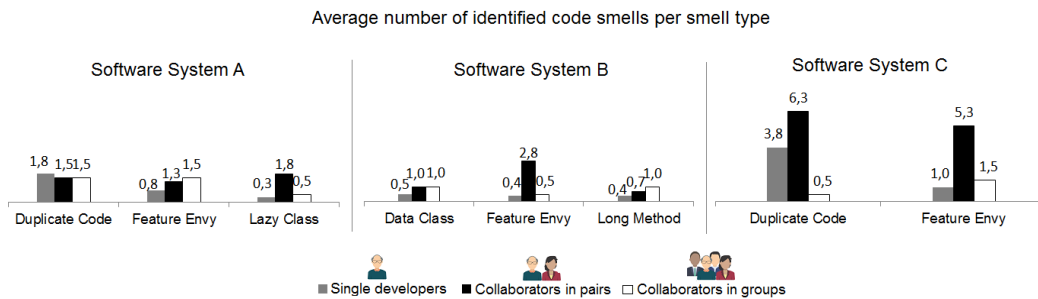


Figure 4.2: Average number of identified code smells per smell type

Data of Figure 4.2 suggest that collaborators achieved a higher average of identified code smells than single developers for nearly all types. In fact, we

observed that collaborators working both in pairs and groups are more effective than single developers when identifying inter-class smells, such as *Feature Envy*, and *Lazy Class*, which affect multiple classes together. In addition, for several inter-class smells, we observed that the average number of code smells identified by pairs or groups was over 40% higher than the average number of identified code smells obtained by single developers.

Through our qualitative analysis, we confirm that the effectiveness of collaborators is a consequence of the inherent complexity of inter-smell types, which require more effort and knowledge of developers to be properly identified. In fact, we observed that the opinion of a single subject was often insufficient to identify inter-class smells precisely. However, collaborators were more precise in such identification because of the knowledge exchange. This observation is reinforced by the opinion of the subjects: 89.28% of all subjects reported that they were more confident on the identification of inter-class smells when working as collaborators. However, some subjects disagree, such as s, who said: “*When we had many contradicting views regarding a code smell, the work in group may lead to considerable difficulties in the decision-making process*”.

*Influential Factor 3.* Collaborators achieved more effectiveness in the identification of code smells, and they achieve a even higher effectiveness in the case of code smells that affect multiple classes.

Finally, we observed that collaborators produced fewer false positives than single developers. In fact, collaborators obtained an average number of false positives equals to 3.0 for pairs and 4.0 for groups, against 4.14 for single developers. Although the difference between the average number of false positives is small between groups and single developers, we observe that pairs are actually more effective and identify almost 25% fewer false positives than single developers. These data reinforce our findings with respect to the average number of code smells. In addition, collaborators showed less optimism when confirming a code smell suspect, because they usually reason more and try more insistently to understand the actual occurrence of a code smell.

Collaborators also showed complementary knowledge when identifying code smells, which led to the lower number of false positives. For instance, subject s1 found a code smell suspect affected by *Feature Envy*, but s2 disagreed by saying that “*...that part of the source code does not have a Feature Envy because the suspect method (marked as containing the code smell) is not calling other methods several times*”. Thus, s2 attempted to prove his opinion by asking s1 to inspect the classes potentially affected by the code smell. Finally, both s1 and s2 refuted the code smell suspect and avoided a false positive.

*Influential Factor 4.* Collaborators very often exchange knowledge, which reduces the number of false positives on the identification of code smells.

#### 4.2.2

##### Collaborative Activities for Identifying Code Smells

Aimed at addressing  $SRQ_3$  of this thesis (Section 1.3), this section discusses the main results of our empirical study aimed at identifying the collaborative activities which most likely contribute to the identification of code smells. Consequently, we address  $RQ_2$  of this chapter as follows.

**$RQ_2$ .** *How do single developers and collaborators perform the identification of code smells?*

To answer  $RQ_2$ , we qualitatively analyzed how collaborators and single developers identify code smells. Based on this analysis, we aimed at characterizing the main collaborative activities which most likely contribute to the identification of code smells. Our goal is to understand the benefits and drawbacks of each collaborative activity.

After transcribing the collected multimedia data (such as video and audio records), we characterized the main activities which compose the identification of code smells. Figure 4.3 summarizes these activities, which are performed by collaborators or single developers, in a notation based on feature modeling (3). Each rectangle represents an activity (feature), which is hierarchically labeled with a number. The type of rectangle border indicates whether the activity was performed by collaborators or single developers, as indicated in the legend. Rectangles with thick borders are activities consistently performed by all the subjects regardless of the session, i.e., whether they were working as single developers, pairs or groups. The figure focuses on presenting the recurring activities performed by at least two third of the subjects. In addition, we mark with (\*) the activities which often contributed to improving the effectiveness of smell identification.

Data of Figure 4.3 suggest that both single developers and collaborators performed six activities, which are usually performed sequentially. These activities, called *phases*, are represented as mandatory features at the top of the model. In feature modeling, mandatory features are those features which always appear in each instance (product) of the model. The six phases are: (1) Smell type selection, (2) Metrics selection, (3) Navigation through the program classes, (4) Identification of smell suspect, (5) Smell suspect validation, and (6) Decision making. Each phase has sub-activities, which are represented by either

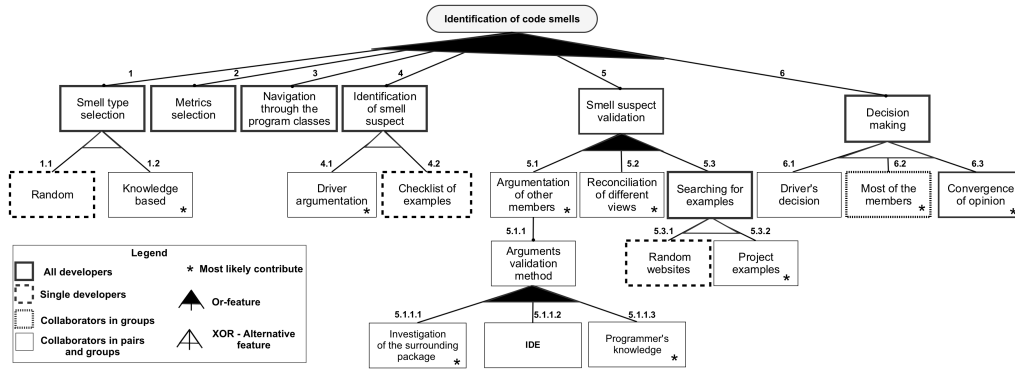


Figure 4.3: Activities on the identification of code smells

the OR (alternative inclusive) or XOR (alternative exclusive) relationships. For instance, the subjects perform the smell type selection (1) either randomly (1.1) or knowledge-based (1.2). Two or more alternative sub-activities may be used together to realize a phase. For instance, pairs or groups in phase 5 may conduct two or three sub-activities (5.1, 5.2, or 5.3) in order to realize the validation of a code smell suspect.

**Phase 1** corresponds to the selection of smell types for identification by developers. We observed that the subjects performed the selection of code smells using random selection (1.1) or a knowledge-based (1.2) approach. In the random selection, subjects selected the smell types by simply following the smell type list presented in the training (Section 4.1.5). In the knowledge-based selection, subjects carefully selected and prioritized the smell types, in accordance with their previous knowledge on certain smell types. For instance, let us consider the following discussion between subjects s8 and s7.

s8 starts the discussion – “What do we have to look for?”

s7 replied – “Let us start with Data Class.”

s8 finally stated – “Right! [Data Class is] a class which stores data only.”

Also regarding Step 1, 71.4% of the single developers conducted the random smell type selection, which suggests that these developers rarely prioritize smell types for identification. We observed that the random selection is less effective than the knowledge-based selection, based on the questionnaire answers and video records. Three key reasons could explain such observation as follows. First, the subjects spent too much time searching for smell types about they know little. Second, as a consequence, they felt discouraged to conduct the identification of code smells. Third, they took too long to start identifying relevant code smells. Finally, we observed that 78.5% of the pairs and 66.67% of the groups conducted the knowledge-based selection, which suggests that pairs are more prone to conducting the knowledge-based selection than groups.



**Phase 2** consists of selecting the software metrics which could support the identification of code smell suspects. These metrics vary according to the smell type. In fact, we observed that the subjects usually used the Number of Lines of Code (LOC) (41) as means to identify occurrences of certain smell types, such as *Long Method* and *Lazy Class*. For instance, let us consider the following discussion between subjects s1 and s2.

s1 starts the discussion – “*What is the next code smell [to identify]?*”

s2 replies – “*Let us look for Long Method.*”

s2 states – “*Okay! We then could first reason about lines [of code].*”

s1 concludes – “*Right! We could start by computing an average number.*”

We also observed that collaborators often exchange knowledge to select the most appropriate software metrics per smell type. Pairs and groups also systematically defined how to understand the results of each metric when identifying a particular smell type. They also identified and discussed potential false positives generated by the use of particular metrics. Finally, collaborators covered a higher number of metrics than single developers.

**Phase 3** consists of developers navigating through the code elements (such as classes and methods) in a software system, aimed at identifying the code smell suspects. We observed that most subjects, regardless of collaborators and single developers, inspect different code elements sequentially in a one-by-one way. On the other hand, we observed that either collaborators or single developers failed to prioritize the code elements for inspection. In the feedback form, the subjects mentioned that this phase could be more effective if all collaborators could navigate through the code elements independently. To illustrate the navigation through the code elements, let us consider the following discussion between subjects s15 and s16.

s15 starts the discussion – “*First, let us navigate through all classes to identify code smell suspects. Maybe we would be more productive this way.*”

(Both subjects debate on the list of smell types)

s15 states – “*Feature Envy is a smell type which uses another class too much. It occurs when the method of a class uses too many attributes of another class*”

s16 completes – “*Let us look at the whole source code and confirm the code smell suspects as we identify them.*”

**Phase 4** consists of identifying code smell suspects. We observe that collaborators usually rely on the driver argumentation (4.1). They elect a collaborator as the *driver*, which leads the discussion among collaborators.

Once all collaborators identify a suspect, the driver argues about it and opens the discussion to the other collaborators. In the case of uncertainty, the collaborators usually end up refuting the suspect. Single developers usually rely on a checklist of examples (4.2). These developers identify a code smell suspect based on a predefined list of smell types, called *checklist of examples*, similarly to the smell type list provided during the experiment. The examples of smell types are randomly selected by the single developers based on subjectivity and knowledge of the developers, which may lead to several false positives.

**Phase 5** consists of validating the code smell suspects, which means confirming or refuting each suspect. We observed that collaborators usually debate on the code smell suspect (5.1). They perform the following alternative activities: the driver navigates through the code (5.1.1.1) to understand the context of a code smell suspect; the collaborators use the functionalities provided by the IDE to navigate through the code elements; or the collaborators discuss about the suspect based on their knowledge (5.1.1.3) acquired by previously inspecting other software systems. We observed that only the use of the IDE functionalities did not contribute to improve the subjects' effectiveness, because IDEs usually lack support to identify code smells.

Also regarding Phase 5, collaborators eventually disagree when confirming a code smell suspect, which requires the intervention of the driver to promote discussion and agreement (5.2). Alternatively, developers may take advantage of examples in their systems or the Web to confirm a suspect (5.3), which is mostly conducted by single developers. We found that single developers usually consult Web search engines, such as Google or Bing (5.3.1). On the other hand, collaborators often consult the system under analysis (5.3.2). Most single developers who adopted the random selection of examples did not properly identify code smells, possibly due to the difference between the inspected code and the code available in other sources. We confirm this observation via screenshots and the list of identified code smells. To illustrate how collaborators exchange knowledge to validate code smell suspects, let us consider the following discussion between subjects s1 to s5.

s2 states – *“This code smell suspect has an Intensive Coupling.”*

s1 disagrees – *“I do not think so.”*

s4 replies – *“Why not? It has several mandatory parameters, which implies several calls to methods that depend on these parameters.”*

s3 reinforces – *“And it [the code smell suspect] could call these methods right here (points out a specific line of code).”*

s4 complements – *“Note that, by changing it [the code smell suspect], you could break the class behavior.”*

s1 states – “Well, I think that it has a Feature Envy. Note that it concentrates several calls, all in a single class. It should be in another class.”

s2 answers – “You are right!”

s5 finalizes – “So, let us confirm the occurrence of a Feature Envy.”

Finally, **Phase 6** consists of making decisions towards the verdict about each code smell. We observed that collaborators usually make decisions based on the driver decision (6.1), which means that the driver leads to the verdict. It makes ineffective the activity whenever collaborators take the driver decision as a sign of authoritarianism. However, the opposite situation occurs whenever the majority opinion of collaborators (6.2) and the agreement of collaborators (6.3) leads to the verdict. The aforementioned discussion between subjects s1 to s5 also illustrates the decision making of Phase 6. The discussion provided in this section leads us to the following finding.

*Finding 1.* Developers perform six activity mandatory during on the identification of code smells: (1) Smell type selection, (2) Metrics selection, (3) Navigation through the program classes, (4) Identification of smell suspect, (5) Smell suspect validation, and (6) Decision making.

We also observed that certain collaborative activities lack tool support aimed at helping developers when identifying code smells; For instance, (5) *Smell suspect validation* consists of validating a code smell suspect. In this case, we observed that developers often need examples of code smell suspects which were validated by developers in their project preferably, in order to support their decision making. However, we did not find a tool that automates this process. In fact, developers that participated in our study often recurred to the Internet to search for examples or randomly into their projects. This result lead us to Finding 2 presented as follows.

*Finding 2.* Some activities, developers have limited support to conduct the collaborative smell identification.

In summary, our results have implications both for organizations and researchers. For instance, organizations could now have empirical knowledge about how to apply the collaborative activities to the identification of code smells, which improves the developers’ effectiveness. Moreover, researchers could propose ways to improve these activities, aimed at improving even more the developers’ effectiveness on the identification of code smells.

### 4.3

#### Threats to validity

The empirical study presented in this chapter shares similar threats to validity of the study presented in Chapter 3, such as the limited inspected smell types and human factors on the creation of the code smell reference list. However, there are additional threats that require discussion. We discuss these threats to validity and minimizations as follows (89).

**Construct Validity.** Our study has a constraint regarding the limited set of inspected smell types, which may have affected our findings. We minimize this possible threat by selecting five well-known and varied smell types. These smell types may affect multiple code elements and are reportedly common in software systems (Section 4.1.3). Another threat regards the elaboration of the code smell reference list, which was performed by two researchers in a manual way with the support of a smell detection tool (52). We mitigate possible threats by engaging researchers that are sufficiently qualified for such elaboration (Section 4.1.4). Regarding varying background of subjects, we mitigate this threat by selecting subjects with minimum knowledge on our topics of interest, such as Java and code smells (Section 4.1.2). We also trained all subjects together to equalize their background. Finally, we adopted a cross-over design (93) in order to reduce biases introduced by the learning of subjects about the study procedures and activities, as well as reduce the problem of our small sample. In fact, we divided the subjects into three groups as discussed in Section 4.1.5.

**Internal Validity.** With respect to the experiment execution, we designed experimental tasks that fit our time constraints of one hour, by relying on experiment simulations performed by volunteers. Through these simulations, we also identified opportunities for encouraging the subjects to participate in the whole experiment. Finally, regarding the fact that subjects did not use a smell detection tool, we highlight that we were not concerned about the effectiveness of the identification of code smells with support of smell detection tools. Instead, we were concerned about how the collaborators identify code smells. Nevertheless, if we had used a tool during the experiment, the results could completely depend on the particularities of a particular tool. Thus, by using a tool, we could have introduced bias to our study data.

**Conclusion Validity.** To conduct the data analysis, we carefully selected the most appropriate statistical tests. We also paid special attention to avoid

violating assumptions of the selected statistical tests. To answer our research question, we applied the Mann-Whitney and Kruskal-Wallis test as discussed in Section 4.1.4. Furthermore, we believe that our questionnaires fit our expectations with the empirical study and support answering our research question. For instance, they allow us to characterize the experienced and inexperienced developers. Thus, we mitigate possible threats related to the analysis of the questionnaires by limiting our analysis to the data gathered through the questionnaires only. Another threat is related to the qualitative data analysis, which may contain biases caused by the viewpoint of the researchers. To mitigate this threat, the qualitative data analysis was performed by multiple researchers together, which validated each others analysis.

**External Validity.** With respect to the generalization of our study findings, our empirical study has some threats, as follows. First, we applied the study only in the context of Brazilian developers, which may not represent any development scenarios around the world due to cultural variations. In addition, even though someone could consider our 28 subjects as a limited set, we did our best to involve the novice developers on the identification of code smells. We minimized possible threats regarding the subjects set by involving the developers with varied background and levels of working experience. We also focused on developers with minimum experience on our topics of interest, such as code smells and pair programming.

#### 4.4 Summary

This chapter aimed at characterizing the influential factors and collaborative activities on the identification of code smells. We conducted a controlled experiment with 28 developers, which have some or none knowledge on the inspected software systems. This chapter compares collaborators and single developers to understand the developers' effectiveness on the identification of code smells. Our results confirm the findings of Chapter 3 and reveal additional details on the collaborative smell identification, which are the influential factors and the collaborative activities. We discuss each finding as follows.

- We have observed that collaborators are more effective than single developers on the identification of code smells, which confirms our previous findings (Chapter 3).
- Collaborators achieved a higher average number of identified code smells than single developers, regardless the smell type. In addition, they are

40% more effective than single developers on the identification of smell types which affect multiple classes (inter-class smells).

- Adding more than two developers to the collaborative smell identification does not necessarily improve the developers' effectiveness of such identification.
- Developers perform several collaborative activities during on the identification of code smells. However, for some activities, developers are not yet properly equipped to conduct the collaborative smell identification.

All the aforementioned findings raise questions about how collaborative activities on the identification of code smells can be improved, particularly the activities for validating code smell suspects. Thus, there is a need to identify what information could support the developers on the identification of code smells. The next chapter addresses this need.

## 5

# An Industrial Multi-Case Study on Collaborative Smell Identification

Our previous studies (Chapters 3 and 4) assessed the effectiveness of collaborative smell identification. They resulted in several findings as follows. First, collaborative smell identification is more effective than individual smell identification. That is, for both novice and professional developers, collaborators identify more code smells than single developers. Second, certain influential factors may improve the effectiveness of smell identification, such as team size and smell granularity. Collaborators are specially more effective than single developers when identifying inter-class smells, which collaborators are often more confident in identifying. Third, certain activities performed by collaborators actually improve the effectiveness of smell identification, mostly due to the exchange of complementary knowledge between the developers. These activities follow a common sequence, from the selection of smell types being considered to the validation of smell suspects.

Although our previous studies led to a number of findings, they mostly focused on developers who have no previous knowledge about the inspected software systems. In addition, the inspected systems were mostly small-sized or legacy systems. As these systems lack real clients and high maintenance demands, they possibly do not have much critical maintenance problems, which may limit the generalization of our observations. Moreover, developers who are familiar with the inspected systems, may help us in revealing even more interesting findings about effective smell identification. In fact, these developers could have a different viewpoint on the negative effects caused by code smells in the maintenance of their systems, which may affect the way developers identify, confirm, or refute the code smell suspects.

Our previous studies suggest the need for improving certain collaborative activities that govern the identification of code smells. However, we have to assess these collaborative activities in real software development contexts. In this case, no developers would be better to engage an empirical study than developers who currently work on their own software systems, in organizations which actually need to save developer efforts due to high software maintenance demands. In fact, an empirical study with these developers could reveal

interesting findings on the collaborative smell identification, which would be hard or impossible to reveal with less experienced developers without previous knowledge about the inspected software systems.

Aimed at addressing the aforementioned limitations, this chapter presents an empirical study which answers the fourth specific research question of this thesis, which states: *SRQ<sub>4</sub>*: **Are there opportunities for improving some collaborative activities associated with smell identification?** For this purpose, we conduct two sessions of an exploratory case study aimed at understanding the effectiveness of collaborative smell identification in industry. Given the intrinsic nature of the case study method, we are mainly concerned here with observing to what extent our previous findings hold in real development settings. We are mostly concerned with the generality of our findings, but without the computation of statistical tests. We discuss our main findings of our industrial multi-case study as follows.

- In industrial settings, we observed that collaborators are also more effective than single developers in the identification of code smells. This observation confirms our previous findings in controlled settings.
- Also in industrial settings, we observed that collaborators often benefited from knowledge exchange to identify certain smell types. Such types are those that require the understanding of multiple code elements (i.e., inter-class smells). This observation also confirms our previous findings.
- In addition to what we found in our previous studies, we observed that developers require several types of information about the local and historical context of a code smell suspect in order to confirm or refute the suspect. To the best of our knowledge, these types of information are often rarely available in existing tool support.

The content of this chapter is an extended version of our work published in the 39th International Conference on Software Engineering (ICSE), Software Engineering in Practice Track (SEIP) (63). The remainder of this section is organized as follows. Section 5.1 describes the study settings. Section 5.2 discusses the effectiveness of collaborative smell identification in industrial settings. Section 5.3 discusses the possible improvements for supporting collaborative smell identification. Section 5.4 discusses threats to validity. Finally, Section 5.5 summarizes this chapter and introduces the next chapter.



## 5.1 Study Settings

We carefully designed our study to investigate the effectiveness of collaborative smell identification in industrial settings. This section presents the study settings, which include the study goal and research questions, the organizations that engaged in the study, and our data sources.

### 5.1.1 Goal and Research Questions

The multi-case study presented in this chapter aimed at understanding the effectiveness of collaborative smell identification. For this purpose, we compare the effectiveness of both collaborators and single developers on the identification of code smells. Differently from our previous studies presented in Chapters 3 and 4, which consist of controlled experiments with small-sized or legacy systems, our case studies focus on both professional developers and software systems from the industry. By relying on the guidelines provided by Wohlin *et al.* (2012), we proposed the following study goal.

- **Analyze** the collaborative smell identification when compared to the individual smell identification,
- **For the purpose of** characterizing the developers' effectiveness,
- **With respect to** precision and recall of identified code smells,
- **From the viewpoint of** professional developers,
- **In the context of** real Java software systems maintained by their own professional developers who participated in the case studies.

The study goal led us to design our first research question as follows.

$RQ_1$ . Are collaborators more effective than single developers in identifying smells in their own industry projects?

With  $RQ_1$ , we aimed at understanding the effectiveness of smell identification from the viewpoint of professional developers who are familiar with the inspected software systems. In order to address  $RQ_1$ , we computed effectiveness similarly to previous chapters: we have considered precision, which measures the correctness of the identified code smells, and recall, which measures the completeness of the identified code smells with respect to all existing code smells in a software system (23) (Chapter 2). Similarly to our previous study, we created a code smell reference list to analyze both precision and recall, which we explain in Section 5.1.4.

In order to better understand how developers identify smells in their own projects, we have created a second research question as follows.

*RQ<sub>2</sub>*. How do professional developers identify code smells in the industry?

With *RQ<sub>2</sub>*, we aimed to deeply understand how professional developers actually identify code smells in certain industry projects. Complementary to *RQ<sub>1</sub>*, this research question targets the different ways in which professional developers conduct the identification of code smells in real development settings, in which developers are mostly concerned about the maintenance of their software systems. We expect that, by addressing *RQ<sub>2</sub>*, we are able to characterize the main benefits and drawbacks of the identification of code smells performed by both collaborators and single developers in the industry.

### 5.1.2

#### Target Software Development Organizations

We selected two Brazilian software development organizations for our case studies. The selection relied on several organization characteristics, such as the experience level of developers with code reviews, the developer team size, and the system domains. Table 5.1 presents these characteristics per organization. The first column lists each characteristic. The second and third column present the data of the two organizations, Org. 1 and Org. 2, respectively. We conducted one case study session with each organization. We observed that both organizations are concerned about identifying poor code structures which decay the maintainability of their systems. Moreover, our first meeting with these organizations showed their excitement to reflect upon the current practices of their developer teams regarding preventive maintenance. In fact, we noticed that one of the two organizations often promotes team training in order to further improve their software maintenance practices. (Appendix I).

Table 5.1: Characteristics per target organization

Characteristic	Organization 1 (Org. 1)	Organization 2 (Org. 2)
Organization type	Public	Private
Number of employees	80	150
Team size per system	3–4	3–7
System type	Information system	Information system
System domain	Government administration	Industrial automation
Programming language	Java	Android/iOS/Java
Platform for inspecting code	-	SonarQube
Code review	Yes	Yes

Together with the organizations, we searched for software systems which are developed in Java, with different sizes, and from different domains. Based

on this search, we selected five software systems for inspection: two systems from Org. 1; and three systems from Org. 2. The selected systems vary from three to seven with respect to the number of developers responsible for maintaining the systems. We asked each system manager to indicate the developers who could participate in our case studies as subjects, from the developers responsible for maintaining the systems. For confidentiality reasons, we are unable to make available the source code of the selected systems.

Table 5.2 summarizes the characterization of subjects who engaged in the two case study sessions, which we refer as CS1 (Org. 1) and CS2 (Org. 2). The first column differentiates the case studies. The second column identified each subject. The third column presents the system inspected by each subject. The fourth column presents the highest education level per subject. Finally, the fifth, sixth, and seventh columns encompass the subject experience with software development, Java, and code review in pairs.

Table 5.2: Subject characterization of CS1 and CS2

Case Study	Subject	System	Education	Software Development (Years)	Java (Years)	Review in pairs (Projects)
CS1	s1	S1	BSc.	7	4	-
	s2	S1	BSc.	7	7	-
	s3	S1	BSc.	9	8	-
	s4	S2	BSc.	9	8	-
	s5	S2	BSc.	12	10	-
	s6	S2	MSc.	5	4	-
	s7	S2	BSc.	10	8	2
CS2	s8	S3	BSc.	12	12	8
	s9	S3	BSc.	13	11	2
	s10	S4	BSc.	4	4	2
	s11	S4	BSc.	8	6	-
	s12	S5	MSc.	4	4	2
	s13	S5	BSc.	5	3	-

**CS1 with a government organization.** Our first case study session was performed with Org. 1, a Brazilian government organization which develops systems for managing budget. Org. 1 recently started to apply code review. We selected two critical software systems (S1 and S2) developed and maintained by the organization for more than seven years. S1 aims at controlling entrances and processing tax revenues of products in the Brazilian state of Amazonas. S2 aims at standardizing budget reports. Any problems in the source code could negatively affect the government accountability and budget. Table 5.2 characterizes the seven subjects (s1 to s7) of CS1 distributed by the software system they maintained at that time. Most subjects are bachelors in Computer Science, having at least four years of professional experience in Java

programming. Only one participant of the case study reported some previous experience in performing peer code review.

**CS2 with a private organization.** Our second case study session was performed with Org. 2, a private non-profit foundation with international customers. The developer teams freely manage the system quality as they wish, but some teams apply code review. We selected three systems (S3, S4, and S5) developed and maintained by Org. 2. S3 supports the management of registry offices of the Amazonas' Court of Justice. S4 manages historical information of patients in a hospital. It uses electronic medical records to integrate patients' clinical and administrative information. S5 traces products in a production line, from their origin to retail locations. Table 5.2 characterizes the six subjects (s8 to s13) of CS2 distributed by the system they maintained at that time. Similarly to CS1, most subjects are bachelors in Computer Science, with three or more years of professional experience in Java programming. However, most subjects had some previous experience with code review in pairs.

### 5.1.3 Data Sources

We carefully selected the following data sources to support our study.

**Data Sources.** To conduct our data analysis, we collected experimental data of the subjects from different data sources, namely: the *subject characterization questionnaire*, the *code smells report questionnaire*, and the *follow-up questionnaire*. We combined the data obtained via these data sources to compensate their strengths and limitations. We describe each data source as follows.

- **Subject characterization questionnaire:** it is composed of questions aimed at characterizing each subject, in terms of their knowledge on topics of interest, such as programming, Java, code smells, and Pair Programming. This material is available at (Appendix D).
- **Code smell report questionnaire:** it is a questionnaire aimed at collecting the list of code smells identified by the subjects during the experimental tasks (available at Appendix E).
- **Follow-up questionnaire:** this questionnaire aimed at collecting the opinion of subjects regarding the identification of code smells conducted in the case studies. This material is available at (Appendix H).

#### 5.1.4

#### Data Analysis Procedures

We defined the following procedures for data analysis.

**Creation of a Code Smell Reference List.** We created a code smell reference list per system. The project manager of each system plus two researchers with experience in identifying smells performed the three following steps. First, the researchers used a smell detection tool to identify code smell suspects. The tool relies on well-known and reportedly effective identification strategies (6, 37). Second, they validated each suspect in isolation. Each involved person then obtained a list of suspects, which could vary since the identification of code smells is subjective. Third, we computed the agreement among the lists obtained by the researchers and system project manager. Both researchers discussed how to solve any conflicts towards a consensus. We also added suspects to the code smell reference list, we discussed their issues, and decided whether it should be added to the reference list, but we confirmed after discussing them with the project manager and other developers.

**Smell Types Inspected in the Case Studies.** to support the generality of our findings, we selected software systems affected by 15 smell types with varying granularity, such as *Long Method*, *God Class*, and *Feature Envy* (29). These smell types affect different code elements, such as methods and classes (29, 37). They are also reportedly very frequent in software systems (91).

**Quantitative Data Analysis.** As aforementioned, to assess the effectiveness of collaborators and single developers, we computed precision and recall. Both measures are calculated based on the number of code smells marked as *true positive* (TP), *false positive* (FP) and *false negative* (FN). Precision and recall are normalized in a range from 0 to 1. High precision values (close to 1) mean that the developer had reported, proportionally, only a few occurrences of FP in the software system. High recall values (close to 1) mean that the developer was able to identify a representative TP number in the software system. Equations 5-1 and 5-2 present the formula for precision and recall, respectively.

$$Precision = \frac{TP}{TP + FP} \quad (5-1)$$

$$Recall = \frac{TP}{TP + FN} \quad (5-2)$$

**Qualitative Data Analysis.** Our qualitative analysis used the following artifacts: the characterization questionnaire, the follow-up questionnaire, screenshots, and multimedia data such as audio and video records. We relied on the procedures of Grounded Theory (GT) (81) to analyze the data, specifically the first and second phases (open coding and axial coding, respectively). Open coding involves the breakdown, analysis, comparison, conceptualization, and the categorization of the data. Axial coding examines the relations between the identified categories. When analyzing the qualitative data, we created codes for the developers' statements (1st phase). Then, these codes were related to each other - through axial coding (2nd phase). Finally, selective coding performs all the process refinements by identifying the core category to which all others are related. We decided not to select a core category herein because a Grounded Theory rule is the circularity between the collection and analysis stages until theoretical saturation is reached (81). Therefore, we decided to postpone the selective coding phase. For this reason, we do not claim that we applied the GT method, only some specific procedures.

We have conducted the open coding on the transcribed data, associating codes with quotations of transcripts, and axial coding, merging and grouping into more abstract categories. For each transcript, the codes and identified networks (memos showing the relationships between the categories) were reviewed, analyzed and changed upon agreement with the other researchers. The phases of the open and axial coding were sufficient to reveal the strategies that developers use to identify code smells as well as the opportunities to improve this task. These procedures allowed us to uncover different strategies followed by developers to identify code smells. We also performed free content analysis over the answers given by the subjects to the follow-up questionnaire.

### 5.1.5 Steps of the Study Execution

Figure 5.1 presents the four steps of our case study. As discussed earlier in this chapter, we conducted a specific study case session per organization: CS1, performed in Org. 1 and CS2, performed in Org. 2. (Section 5.1.2). To conduct the case study, we asked all subjects to first fill out and sign a consent questionnaire. The subjects then started to participate in the actual smell identification tasks of the study. We describe each experimental step as follows.

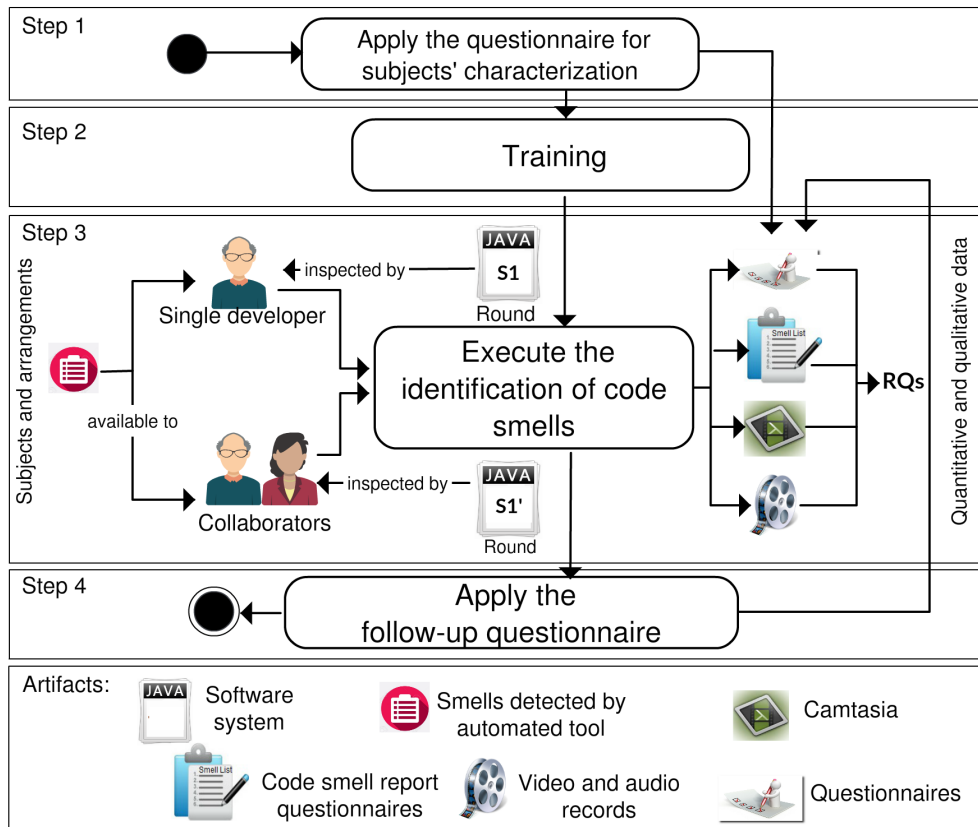


Figure 5.1: The experimental design

**Step 1. Apply the Subject Characterization Questionnaire.** The subject characterization questionnaire aims at characterizing each subject who participated in the experiment. The questionnaire includes questions regarding the background of the subject in formal education, software development, the Java programming language, and peer review. The responses obtained through this questionnaire allowed us to identify some key characteristics of each subject, as presented in Section 5.1.2.

**Step 2. Training of Subjects.** After characterizing the subjects, we provided them with a training session. This training aimed at supporting subjects to properly understand and execute the experiment (Appendix J). The training was organized in two parts. First, during 25 minutes, we explained the technical concepts and terminologies related to this study. Second, we took 10 minutes to conduct a discussion about the technical concepts and terminologies.

**Step 3. Smell Identification Task.** Per experiment session, we asked the subjects to engage in two rounds of code smell identification in a cross-over design fashion. They worked on the same system in both rounds, but in disjoint

sets of modules in each round. Figure 5.1 presents the identification task. In the first round, the developers performed individual smell identification in a set of modules (e.g., S1 in the Figure 5.1). In the second round, they performed collaborative smell identification, but in other set of modules. (e.g., S1' in the Figure 5.1). In addition, in each round, the subjects were asked to annotate the identified smells in the code smell report questionnaire. Table 5.3 presents the arrangements of subjects according to their systems. The subjects received a list of code smells from 15 different types detected by an automated tool to be used as a baseline (Section 5.1.4). In this sense, subjects were instructed to feel free to follow or not the given list. Subjects also received a guide that characterizes each type of code smell used in this study. Each round lasted 45 minutes. At the end of the experiment, each subject answered a follow-up questionnaire and provided a list of identified smells.

Table 5.3: Subject arrangement

CS1		
Systems	Single developers	Collaborators
S1	s1, s2, s3	s1, s2 and s3
S2	s4, s5, s6 and s7	s4 and s5 s6 and s7
CS2		
Systems	Collaborators	Single developers
S3	s8 and s9	s8, s9
S4	s10 and s11	s10, s11
S5	s12 and s13	s12, s13

**Step 4. Answer the follow-up questionnaire.** After participating in the two sessions of the experiment, the participants filled a follow-up questionnaire. This questionnaire aimed at collecting the impressions of subjects regarding the experiment. We aimed at understanding their opinion about the identification of code smells and the experience of working collaboratively to identify code smells. More details about this step are provided in (Appendix H).

## 5.2

### Effectiveness of Collaborative Smell Identification in Industry

This section presents the results of the quantitative data for both collaborations and single developers, per case study session, as follows.

**Results for CS1.** Table 5.4 shows the results of the first round in CS1. The table shows the TP, FP, and FN measures associated with each single developer. Each developer, represented by his ID (2nd column), worked on a specific system (1st column). The last columns present the precision and recall



results. Table 5.5 follows a similar structure. However, it presents the results of the second round in the same organization. Thus, the 2nd column shows the collaborators. The four collaborators, who worked on the S2 system in round 1, were divided in two pairs in round 2. The three collaborators of the S1 system worked together as a group in round 2.

Table 5.4: Precision and recall of single developers in CS1

System	Subject	TP	FP	FN	Precision	Recall
S1	s1	3	3	52	0.50	0.05
	s2	7	3	48	0.70	0.13
	s3	9	6	46	0.60	0.16
	s4	8	5	50	0.62	0.14
	s5	5	2	53	0.71	0.09
S2	s6	5	3	53	0.63	0.09
	s7	2	1	56	0.67	0.03

Table 5.5: Precision and recall for collaborators in CS1

System	Subject	TP	FP	FN	Precision	Recall
S1	s1, s2 and s3	15	8	40	0.65	0.27
	s4 and s5	9	3	49	0.75	0.16
S2	s6 and s7	7	0	51	1.00	0.12

Comparing precision and recall of Tables 5.4 and 5.5, we observe the following trend: precision was consistently improved (except for s2) in round 2, while recall was slightly improved. We did not expect any improvement on recall given the time constraints. The three collaborators, who worked as a group, did not achieve clearly better results than when conducted the identification of code smells as single developers, except for recall.

**Results for CS2.** Differently from the case of CS1, developers from CS2 worked as collaborators in round 1 and as single developers in round 2. Table 5.6 and 5.7 present the results achieved by each group and each single developer, respectively. According to these results, precision and recall were consistently better (except for s12) for collaborators than for single developers. s12 achieved similar results in both rounds.

Table 5.6: Precision and recall for collaborators in CS2

System	Subject	TP	FP	FN	Precision	Recall
S3	s8 and s9	6	0	29	1.00	0.17
S4	s10 and s11	18	2	3	0.90	0.86
S5	s12 and s13	3	3	27	0.50	0.10

Table 5.7: Precision and recall for single developers in CS2

System	Subject	TP	FP	FN	Precision	Recall
S3	s8	5	11	30	0.31	0.14
	s9	5	14	30	0.26	0.14
S4	s10	13	5	8	0.72	0.62
	s11	5	8	16	0.38	0.24
S5	s12	3	8	27	0.50	0.10
	s13	2	7	28	0.22	0.07

**Effects of Collaboration on the Identification of Code Smells.** We compare the results for single developers and collaborators in order to answer RQ1: *Are collaborators more effective than single developers in identifying smells in their own industry projects?* As explained in 5.1.4, we used precision and recall to compare the developers' effectiveness in both situations. After analyzing the results of the two organizations (Tables 5.4, 5.5, 5.6, and 5.7), we noticed some similar trends in both case study sessions.

First, almost all developers of both organizations reached better precision and recall for collaborators and single developers. Only one developer (s2) from the Org. 1 achieved better precision as single developer. We did not observe any effect of swapping the order of single developers and collaborators along the two case study sessions. In other words, the collaborative smell identification outperformed the individual smell identification in both sessions.

Second, we observed that the Org. 2 developers reached better precision and recall results as collaborators than the Org. 1 developers. By analyzing the characterization form (Section 5.1.2), we noticed that Org. 2 developers had previous experience with code review in pairs, while Org. 1 developers had none. This experience likely helped them to better explore the benefits of collaborative smell identification. These results lead us to our first finding:

Finally, as far as recall is concerned, we noticed that the results were far from 1.0 (100%) in general. This behavior was somehow expected given usual time constraints of the case study sessions. In fact, we have noticed that developers clearly tended to focus only on smell types that they consider as priority for identifying and eliminating. However, we observed that single developers produced fewer FNs only because they tended to identify much simpler smell types (Section 5.3) when working in isolation. In contrast, collaborators were able to identify more complex smell types (Section 5.3), which remained unnoticed by single developers. Moreover, developers avoided making mistakes (FP) when working with somebody else. This was confirmed by the analysis of the qualitative data.

*Finding 1.* Collaborators tend to be more effective than single developers on the identification of code smells.

### 5.3

#### Improving Collaborative Smell Identification

Analyzing all discussions among participants and based on the quantitative analysis, besides the answers obtained from the follow-up questionnaire, we could answer our second research question: *How do professional developers identify code smells?* We performed a qualitative analysis to identify the developers' actions performed by developers in both rounds. Moreover, we checked whether those developers' actions contributed to the effectiveness of the identification of code smells (TP) or otherwise (FP and FN).

**Code Smells and Comfort Zone.** As presented in Section 5.2, when we analyzed the code smells identified by the developers, no one found the same code smell in the same class. There was no intersection of code smells reported by two or more developers. It can be explained by the observed trend of developers focusing their analysis on the code which they had worked in the past. In fact, we noticed that single developers tend to stay in their comfort zone concerning the analyzed classes. In most of the cases, they analyzed only the classes that they knew. As a result, they often identified simpler smell types internal to the class under their ownership, such as *Long Method*.

**Leveraging Complementary Knowledge.** On the other hand, when developers worked as collaborators, they were keen to analyze other classes indicate by each other. As soon as they shared their knowledge about different classes, they started to reveal more complex smell types affecting multiple classes. For example, s1, s2 and s3 were analyzing ClassA during the search of a *Long Parameter List* in CS1. When they were analyzing ClassA, s1 reported that a part of the code in that class was duplicated from ClassB. As s1 noticed the duplication, they confirmed these classes were embodying *Duplicated Code*. Therefore, s1 shared his knowledge about ClassB with the other two developers. This situation is reported below:

s1 – “Guys, let us go first to ClassB. Methods in the ClassB are duplicated”

s3 – “No, man.”

s1 – “They are the same”

s3 – “Hold on... which methods?”

s1 – “The *ClassA.setY* and *ClassB.setZ*”

Developer s3 opens the *ClassB*.

s3 – “Which method?”

s1 – “It’s the second one”

s3 – “is this the method?”

Developer s1 points to the *ClassB.setZ* method

s1 – “Now, go to the *ClassB.setZ* method... its core is duplicated. This piece of code is the same from *ClassA.setY*.”

s3 – “Jeez. it’s the same. I had no idea about it.”

This example illustrates a scenario in which collaboration helped the developers to identify a code smell suspect in code elements that they did not know. A developer may not know about the entire system, but when he teams up with another developer, they can benefit from the individual knowledge of each other. Thus, they can identify code smells that require understanding multiple code elements of the system like exemplified. During the analysis, we found more cases of code smells that were identified only when the developers teamed up. Most cases are related to *Duplicated Code*, *God Class*, *Shotgun Surgery*, *Refused Bequest*, and *Speculative Generality*. These smell types often require a non-local knowledge of the system. We also observed that collaborators could also eliminate false positives (related to these smells) yielded by the smell detection tool. As single developers, developers did not have sufficient knowledge to refute the tool outcomes and sometimes accepted them. This result leads us to our second finding:

*Finding 2. Collaborators benefited from knowledge exchange to identify code smells that require a broad understanding of the system.*

The identification of the *God Class* smell type required developers to inspect if each code smell suspect fulfills more than one responsibility. For example, developers s1, s2, and s3 were investigating the *InvoiceInput* class. They were discussing whether the class was a *God Class* or not. They mentioned that in their systems, the report generation is related to several classes – no class is responsible for generating all reports. Otherwise, such class would be a *God Class*. Then, they realized that the *ReportGenerationAction* class was generating different types of reports. Thus, the developers indicated that the *ReportGenerationAction* class was a *God Class*.

s2 – “But if we select this (*InputInvoice*) class, you have to start with *God Class* since this class has all the types of reports. Those total (attributes)

*are all from here.”*

s3 – *“Yeah, God Class and Feature Envy smells.”*

s1 – *“Yeah, but I wonder if this class is a God Class, what if the report (generation) is related to the InvoiceInput class?”*

s2 – *“Yeah, the issue we have here is that we won’t have a report (generation) that is related to only one class. It (the report generation) always includes much others (classes). For example, the tax, the invoice, the item, What should we do?”*

s1 – *“Do we need to create a separate class for that?”*

s2 – *“If we create a class for report (generation)... but I’m not sure.”*

s3 – *“By the way, there is a class in charge of generating reports.”*

Developers open the ReportGenerationAction class

s1 – *“ReportGenerationAction.”*

s2 – *“No, but it (the class) is a action, it is not a DAO.”*

s1 – *“And it is not being considered as a class.”*

s3 – *“Yeah, but look, this class is a God Class. Besides God Class and Feature Envy, what else do we have?”*

*God Class* and *Duplicated Code* seem to be smell types which require a broader knowledge of the software system. Sometimes, developers need to reason about multiple code elements before confirming whether the element has a code smell or not. Thus, given the need for global reasoning, knowledge exchange among two or more developers may help them to better identify particular smell types. Therefore, the number of developers involved in identifying of code smells may affect the developers’ effectiveness.

**Contextual Information.** Finally, we noticed that collaborators needed contextual information to make well-informed decisions about each code smell suspect. The *God Class* case shows that the identification of code smells is not limited to analyzing a set of metrics and thresholds related to the code smell suspect. Developers had to verify whether the class fulfilled more than one responsibility. The understanding of each responsibility may require the analysis of various classes realizing that responsibility. In other words, developers needed to analyze a wide range of contextual information to properly identify code smells, leading to our third finding:

*Finding 3. Developers needed various types of contextual information before making a decision about the code smell.*

After, we classified the contextual information frequently mentioned by developers in four categories, which are presented below.

**Surrounding Information.** In the case of intra-class smells, developers need to analyze the code elements that surround the class that contains a code smell. For example, to confirm a *Feature Envy*, it is also necessary to inform which methods and attributes the envy method accesses in another class. Also, developers needed to check whether the class in which the method seems to be interested should receive the envious method or not. Moving the method would also have other implications for the clients of the target class. By analyzing the video records of the identification performed by developers, we observed that the developers often had to simulate the refactoring in order to reflect upon advantages and disadvantages before making a decision.

**Historical Information.** The developers used the historical information of a code element to understand its evolution. They often tried to understand what happened with the code element across different versions of the system since its creation. For example, they were trying to figure out what happened with a class that used to be affected by a method-level code smell but which now is affected by a class-level smell. The thorough understanding of the code smell history could help indicate whether the suspect was indeed a code smell or not. For instance, developers tried to reveal through the history when the class became a *God Class*, i.e., when it started to implement other functionalities:

s8 – “*It became a complex class over the time since it was used for several other things, including various other types for verification. Formerly, it was only a class used to verification, but then it became integrated into many other classes as quota request and debt recognition. Today, it is also used to load a part of the pledge. Thus, it began to serve various system clients, starting to be very complex (...)*”

**Developer Knowledge.** Developers may use the information provided by other developers who contributed to the implementation of each class. Thus, they can exchange knowledge about the classes in order to confirm or refute the existence of a code smell. In the following, we present a situation in which the developer’s knowledge about the class was essential to avoid a false positive involving a *Duplicated Code*. In this case, developers s2 and s3 did not know much about the inspected class. However, s1 helped to implement the class. Thus, he used his knowledge of the method to explain why it should not be considered as a *Duplicated Code*.

- s2 – *“This method is so long. I can’t understand what it is doing.”*
- s1 – *“Here’s the thing... this method has two implementations. The first one uses a rule, but the rule changed after a certain data. So, all the entities created before the data use the first implementation of the method, and all the entities created after the data use the new implementation of the method. We have this problem; the code has a temporal rule. This is crazy.”*
- s3 – *“So it’s not considered a Duplicated Code”*
- s1 – *“No. It’s not a Duplicated Code”*

**Framework Information.** Developers also need information about the frameworks used in the system. This information may help the developers to understand why the code element (associated with the use of a framework or a library) has a smell or not. If the developer knows about the used framework (or library), he can configure the smell detection tool. Thus, he can avoid false positives generated because the developer used a framework. For example, during the identification of code smells, s8 and s9 noticed the importance of being aware of the framework. In this case, s8 mentioned that identifying code smells with support of a smell detection tool is not an easy task because there are some code smells that require understanding the context of the code element. He mentioned that the code element should not be considered as having a smell because the framework forced the developer to implement in that way.

- s8 – *“It (the class) is doing what it supposed to do. It is sending what it supposed to send.”*
- s9 – *“But the reason (of the code smell) here is the framework (...)”*
- s8 – *“This element is hard to detect correctly because of the framework. We have to understand the context here. The reviewer needs to understand the context.”*

## 5.4

### Threats to Validity

**Construct validity.** We highlight the following threats to construct validity concerning the case study plan: (i) the distribution of smells per project, (ii) the composition of the smell reference list, and (iii) the small subject sample. We mitigate (i) by selecting classes of each project composed by similar smell distributions. We mitigate (ii) by composing the reference list with the support of two researchers and software managers per project. The set of 15 code smells used in the study covered smells at different levels of granularity. Moreover, there is empirical evidence that such code smells are

associated with varying degrees of maintenance effort (Section 5.1.4). Finally, we mitigate the biases caused by (iii) by adopting a cross-over design (93).

**Internal Validity.** The time restriction to conduct the identification of code smells can be considered a threat. We estimated that the developers would have a chance to finish the identification task. This estimation was the result of a pilot phase that we a before the study. Based on the experience of the pilot phase and on the time constraints of the organizations, we calculated that the time limit of 45 minutes would be sufficient to participants identify a considerable list of code smells. It is noteworthy that the pilot also allowed us to identify opportunities for improving the study design.

**External Validity.** Although it is expected from case studies to observe “in the small,” the execution of different instances of the same case studies can be useful to strengthen the findings. The limited diversity of contexts involved in the case studies can be considered a threat to validity. However, we argue that the selected organizations represent typical software development organizations in Brazil and elsewhere. We described their profile in detail, thus others can understand the contexts related to both studies: CS1 and CS2.

**Conclusion Validity.** In order to mitigate threats regarding the conclusion validity, we planned different methods and instruments for collecting both quantitative and qualitative data. Consequently, it was possible to triangulate evidence that emerged from the practice, researchers’ observations, and participants’ opinion, strengthening the study findings.

## 5.5 Summary

In this chapter, we investigated the impact of collaboration on the effectiveness of the identification of code smells. Differently from the previous chapters, we ran a multi-case study involving five software projects and 13 professional developers from two software development organizations. These developers were asked to identify code smells in their own software projects. In order to answers our two complementary research questions, we performed quantitative and qualitative analyses. The results suggest that developers working collaboratively on the identification of code smells tend to be more effective than developers working individually. In fact, when developers worked collaboratively, they benefited from the shared knowledge to identify code smells that require a broad understanding of the system.



In addition, our results suggest that organizations should encourage collaboration between developers to increase the rate of success on the identification of code smells. Moreover, the results also indicate that the collaboration helped developers to reduce the number of mistakenly identified code smells during the identification task. This finding indicates that collaboration can be used to save effort on considering opportunities of refactoring that are not cost-effective to the project. Finally, analyzing the qualitative data, we also noticed some contextual information that the developers need before making a decision about the code smell. Examples of contextual information include the surrounding context of the affected element, the historical information of a element, and the developer information.

Collaboration has been shown useful to improve the developers' effectiveness in different software engineering activities, such as code review (7, 48). However, none of these previous studies applies and investigates the use of collaboration in the context of the identification of code smells. As discussed in this thesis, several smell types are perceived as critical for identification and elimination by professional developers. By discussing previous work, we speculate that collaboration has a potential to improve the identification of code smells, due to the subjective nature and the inherent difficulties of this task.

However, as previously discussed, there is limited empirical knowledge about the effectiveness of collaborative smell identification. In fact, although collaboration may improve the developers' effectiveness in identifying code smells, we lack studies that compare the effectiveness of collaborators and single developers (63). Overall, organizations have little knowledge on how to adopt collaboration to improve the developers' effectiveness in the identification of code smells (43). Moreover, organizations know little about how to properly conduct the identification of code smells, as well as what influential factors could increase the number of identified code smells and decrease the number of misidentified code smells. Thus, without empirically-grounded guidance, they are likely to not adopt collaborative smell identification.

To address the aforementioned limitations, this thesis presents several complementary studies to understand the influence of developer's collaboration on smell identification effectiveness. At first, we conducted controlled experiments to assess whether collaborators are more effective than single developers. Our data analysis relies on the analysis of small-sized or legacy software systems for inspection and identification of code smells, in addition to novice and professional developers without previous knowledge about the inspected systems. Next, we conducted two sessions of a case study aimed at reinforcing our previous findings in real development settings. We rely on the analysis of systems with real clients and high maintenance demands, in addition to professional developers who have maintained the inspected systems.

## 6.1

### Main Findings

Table 6.1 summarizes the main findings obtained through this thesis. The first column presents the purpose of each finding, which corresponds to the issue which the finding addresses. The second column presents the findings as they are described in their respective chapter of the thesis. In summary, our results have implications both for practitioners and researchers. For instance, organizations can now explore empirical knowledge about collaborative smell identification in order to revisit their code review practices. In turn, researchers could use these findings to support future research on the collaborative smell identification, which we suggest in Section 6.3.

Table 6.1: Summary of study findings of this thesis

Purpose	Study Finding
Effectiveness	Collaborators tend to be more effective than single developers in the identification of code smells, regardless of their working experience. In fact, both average precision and recall have improved with collaborative smell identification.
	In industrial settings, we observed that collaborators are more effective than single developers in the identification of code smells. This observation confirms our previous findings in controlled settings.
	Collaborators achieved higher effectiveness in the identification of inter-class smells, i.e., those smells that affect multiple classes. Indeed, collaborators identified a higher average number of code smells because they almost always had to consider both developers' knowledge on revealing scattered, complementary symptoms associated with a single smell type.
Influential Factors	Collaborators benefit from information exchange during smell identification. Consequently, they are able to correctly identify several code smells and significantly reduce the number of misidentified code smells.
	We also derive empirical evidence that bringing together more than two developers to the task of collaborative smell identification does not necessarily improve their effectiveness.
	Collaborators benefit from information exchange during the identification of code smells. Consequently, they are able to correctly identify several code smells and reduce the number of misidentified code smells.
	Also in industrial settings, we observed that collaborators often benefited from knowledge exchange to identify certain smell types, which require the understanding of multiple code elements (inter-class smells). This observation also confirms our previous findings.
Activities	Developers perform several collaborative activities during the identification of code smells.
	However, for some activities, developers have limited support to conduct collaborative smell identification.
Improvements	Additionally to our previous studies, we observed that developers require several types of contextual information about a code smell suspect in order to confirm or refute the suspect (Chapter 5.5). For instance, these contextual information include surrounding elements and history of the smelly structure and framework-induced smells.

## 6.2

### Recommendations

By relying on the findings obtained through this thesis, which we summarized in Table 6.1, we conclude that collaborators are more effective than single developers on the identification of code smells. Overall, we observed that collaborators achieve higher precision than single developers, which implies a lower number of false positives. Consequently, collaborative smell identification may reduce the maintenance effort, including the number of unnecessary changes performed by developers towards the elimination of false positives. This observation leads us to our first recommendation.

*Recommendation 1.* Organizations should encourage collaborative smell identification in their teams to better reveal critical refactoring opportunities and ignore unnecessary ones.

In addition, we observe that developers usually need to reason about multiple code elements of the software system before confirming or refuting a code smell suspect. Thus, the knowledge exchange among collaborators is essential to help developers in identifying particular smell types in a more effective way. This observation leads us to our second recommendation.

*Recommendation 2.* Organizations should promote developer collaboration for improving the identification of code smells, mostly when it requires reasoning about multiple code elements (inter-class smells).

Finally, we noticed that developer teams often need contextual information to confirm or refute each code smell suspect. In fact, for certain smell types, the developers analyzed various characteristics of the code elements of the software system. In other words, developers needed to analyze a wide range of contextual information to properly identify code smells. However, existing tool support for smell detection neglects these types of contextual information. This observation leads us to our third recommendation.

*Recommendation 3.* Researchers could propose means to support collaborative smell identification by providing useful contextual information to developers, which potentially helps confirm or refute code smell suspects.

### 6.3

#### Future Work

This thesis resulted in multiple findings on the effectiveness of collaborative smell identification. Our study addresses limitations of the literature about the role of collaboration specifically in the context of the identification of code smells. Based on a combination of controlled experiments and case studies, we were able to understand to what extent collaborators are more effective than single developers. We also identified influential factors which potentially improve the developers' effectiveness on smell identification. In addition, we characterized the main collaborative activities performed alongside smell identification. Finally, we revealed some opportunities for improving these activities which improves the effectiveness of collaborative smell identification. However, this thesis does not provide a definitive body of knowledge about collaborative smell identification. There are several extensions to our research, which leads us to the following suggestions for future work.

**Additional empirical studies in industrial settings.** In fact, we conducted two sessions of a case study with two Brazilian development organizations, which are mostly focused on building data management software systems. Thus, future work could explore other system domains as well as other organizations with different settings and from different countries. Due to the subjective nature of code smell identification, both domain, organizational and cultural factors could affect the developers' effectiveness. Consequently, there are many possibilities to replicate our case studies in the industry.

**Systematic conduction of collaborative activities.** We have revealed that smell identification is usually much more complex than it is usually assumed or advertised. Our studies, both controlled experiments and case studies, revealed several activities that are typically performed alongside smell identification. Specifically in the case studies with professional developers, we identified additional collaborative activities performed by these developers but not identified in our controlled studies with novice and professional developers. Thus, there is a need for thoroughly evaluating the impact of each activity in order to understand its effects on the developers' effectiveness and other important aspects, such as efficiency, motivation and productivity.

**Adoption of collaborative smell identification from the start.** This thesis provides some recommendations for organizations to adopt collaborative smell identification in their settings, thereby aiming at improving the

effectiveness of smell identification in practice. However, although we have conducted case studies with organizations and professional developers, we did not evaluate to what extent our recommendations actually improve the developers' effectiveness in organizations which are adopting collaborative smell identification from the start. Consequently, an evaluation of the adoption of collaborative smell identification from the start is valuable.

**Leveraging contextual information.** In our studies, we have observed that developers often need to understand the surrounding context of a code smell suspect, in order to confirm or refute the actual occurrence of a code smell. However, we were not concerned with algorithms and strategies to extract and represent this information, making it explicitly available to developers. In addition, we did not conduct empirical studies aimed at systematically evaluating how each type of contextual information could improve developers' effectiveness on smell identification. In this context, several empirical studies could be conducted with both novice and professional developers, aimed at verifying to what extent each type of context information actually improves the developers' effectiveness in different organizations.

**Tooling support for collaborative smell identification.** In this thesis, we present several findings that can base the proposal of novel tooling support for collaborative smell identification. In particular, we concluded that there is a need for tool features for explicitly supporting developers through the confirmation or refutation of code smell suspects. For instance, one could propose a tool that visually represents contextual information to help developers in confirming or refuting a code smell suspect. We did not find tools with such features either in the literature or in industry.

## Bibliography

- [1] ABBES, M.; KHOMH, F.; GUÉHÉNEUC, Y. ; ANTONIOL, G.. **An empirical study of the impact of two antipatterns, blob and spaghetti code, on program comprehension.** In: Proceedings of the 15th European Conference on Software Maintenance and Reengineering (CSMR), p. 181 – 190, 2011.
- [2] ABRAHAMSSON, P.; BABAR, M. A. ; KRUCHTEN, P.. **Agility and architecture: Can they coexist?** IEEE Software, 27(2):16–22, 2010.
- [3] APEL, S.; BATORY, D.; KÄSTNER, C. ; SAAKE, G.. **Feature-Oriented Software Product Lines.** Springer, 2013.
- [4] ARISHOLM, E.; GALLIS, H.; DYBA, T. ; SJOBERG, D. I.. **Evaluating pair programming with respect to system complexity and programmer expertise.** IEEE Transactions on Software Engineering, 33(2):65–86, 2007.
- [5] BACCHELLI, A.; BIRD, C.. **Expectations, outcomes, and challenges of modern code review.** In: PROCEEDINGS OF THE 2013 INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING, ICSE '13, p. 712–721, San Francisco, CA, USA, 2013. IEEE Press.
- [6] BAVOTA, G.; LUCIA, A. D.; PENTA, M. D.; OLIVETO, R. ; PALOMBA, F.. **An experimental investigation on the innate relationship between quality and refactoring.** Journal of Systems and Software, 107(C):1 – 14, 2015.
- [7] BAYSAL, O.; KONONENKO, O.; HOLMES, R. ; GODFREY, M. W.. **Investigating technical and non-technical factors influencing modern code review.** Empirical Software Engineering, 21(3):932–959, 2016.
- [8] BEGEL, A.; NAGAPPAN, N.. **Pair programming: what’s in it for me?** In: PROCEEDINGS OF THE SECOND ACM-IEEE INTERNATIONAL SYMPOSIUM ON EMPIRICAL SOFTWARE ENGINEERING AND MEASUREMENT, ESEM'08, p. 120–128, New York, NY, USA, October 2008. ACM.

- [9] BEGEL, A.; SIMON, B.. **Novice software developers, all over again.** In: PROCEEDINGS OF THE FOURTH INTERNATIONAL WORKSHOP ON COMPUTING EDUCATION RESEARCH, ICER '08, p. 3–14, New York, NY, USA, 2008.
- [10] BELLER, M.; BACCHELLI, A.; ZAIDMAN, A. ; JUERGENS, E.. **Modern code reviews in open-source projects: Which problems do they fix?** In: PROCEEDINGS OF THE 11TH WORKING CONFERENCE ON MINING SOFTWARE REPOSITORIES, MSR 2014, p. 202–211, New York, NY, USA, 2014. ACM.
- [11] BELLON, S.; KOSCHKE, R.; ANTONIOL, G.; KRINKE, J. ; MERLO, E.. **Comparison and evaluation of clone detection tools.** IEEE Transactions on software engineering, 33(9):577–591, 2007.
- [12] BENNETT, K. H.; RAJLICH, V. T.. **Software maintenance and evolution: A roadmap.** In: PROCEEDINGS OF THE CONFERENCE ON THE FUTURE OF SOFTWARE ENGINEERING, ICSE '00, p. 73–87, New York, NY, USA, 2000. ACM.
- [13] BOEHM, B.; BASILI, V. R.. **Software defect reduction top 10 list.** Computer, 34(1):135–137, 2001.
- [14] BOOCH, G.; RUMBAUGH, J. ; JACOBSON, I.. **Unified Modeling Language User Guide, The (2Nd Edition) (Addison-Wesley Object Technology Series).** Addison-Wesley Professional, 2005.
- [15] GRANT BRAUGHT, J. M.; WAHL, T.. **The benefits of pairing by ability.** In: PROCEEDINGS OF THE 41ST ACM TECHNICAL SYMPOSIUM ON COMPUTER SCIENCE EDUCATION, SIGCSE '10, p. 249–253, New York, NY, USA, 2010. ACM.
- [16] BROWN, W. H.; MALVEAU, R. C.; MCCORMICK, H. W. S. ; MOWBRAY, T. J.. **AntiPatterns: Refactoring Software, Architectures, and Projects in Crisis.** John Wiley & Sons, Inc., New York, NY, USA, 1st edition, 1998.
- [17] BRYTON, S.; BRITO, F.; ABREU ; MONTEIRO, M. P.. **Reducing subjectivity in code smells detection: Experimenting with the long method.** In: 7TH INTERNATIONAL CONFERENCE ON THE QUALITY OF INFORMATION AND COMMUNICATIONS TECHNOLOGY, p. 337–342. IEEE Computer Society, 2010.



- [18] CARNEIRO, G.; SILVA, M.; MARA, L.; FIGUEIREDO, E.; SANT'ANNA, C.; GARCIA, A. ; MENDONÇA, M.. **Identifying code smells with multiple concern views**. In: 10TH BRAZILIAN SYMPOSIUM ON SOFTWARE ENGINEERING, p. 128–137, Salvador, Brazil, 2010. IEEE.
- [19] D'AMBROS, M.; BACCHELLI, A. ; LANZA, M.. **On the impact of design flaws on software defects**. In: PROCEEDINGS OF THE 2010 10TH INTERNATIONAL CONFERENCE ON QUALITY SOFTWARE, QSIC '10, p. 23–31, Washington, DC, USA, 2010. IEEE Computer Society.
- [20] DEITEL, H. M.; DEITEL, P. J.. **Java How to Program (6th Edition)**. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 2004.
- [21] DA SILVA ESTÁCIO, B. J.; PRIKLADNICKI, R.; DA COSTA MÓRA, M.; NOTARI, G.; CAROLI, P. ; OLCHIK, A.. **Software kaizen: Using agile to form high-perfomance software development teams**. In: Lacey, M., editor, AGILE CONFERENCE 1, 2014, AGILE, p. 1–10, Orlando, FL, USA, 2014. IEEE.
- [22] ESTACIO, B.; OLIVEIRA, R.; MARCZAK, S.; KALINOWSKI, M.; GARCIA, A.; PRIKLADNICKI, R. ; LUCENA, C.. **Evaluating collaborative practices in acquiring programming skills: Findings of a controlled experiment**. In: SOFTWARE ENGINEERING (SBES), 2015 29TH BRAZILIAN SYMPOSIUM ON, p. 150–159, Sept 2015.
- [23] FAWCETT, T.. **An introduction to roc analysis**. Pattern recognition letters, 27(8):861–874, 2006.
- [24] FERNANDES, E.; OLIVEIRA, J.; VALE, G.; PAIVA, T. ; FIGUEIREDO, E.. **A review-based comparative study of bad smell detection tools**. In: PROCEEDINGS OF THE 20TH INTERNATIONAL CONFERENCE ON EVALUATION AND ASSESSMENT IN SOFTWARE ENGINEERING, EASE '16, p. 18:1–18:12, Limerick, Ireland, 2016. ACM.
- [25] FERNANDES, E.; SOUZA, P.; FERREIRA, K.; BIGONHA, M. ; FIGUEIREDO, E.. **Detection strategies for modularity anomalies: An evaluation with software product lines**. In: INFORMATION TECHNOLOGY-NEW GENERATIONS, p. 565–570. Springer, 2017.
- [26] FERNANDES, E.; FERREIRA, L. P.; FIGUEIREDO, E. ; VALENTE., M. T.. **How clear is your code? an empirical study with programming challenges**. In: CONFERENCE: 20TH IBERO-AMERICAN CONFERENCE ON SOFTWARE ENGINEERING, ESELAW TRACK, CO-LOCATED WITH

- 39TH INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING (ICSE), ESELAW '17, p. 1–10, Buenos Aires, Argentina, 2017.
- [27] FERREIRA, M.; BARBOSA, E. A.; BERTRAN, I. M.; ARCOVERDE, R. ; GARCIA, A.. **Detecting architecturally-relevant code anomalies: a case study of effectiveness and effort.** In: SAC'14, p. 1158–1163, 2014.
- [28] FONTANA, F. A.; MARIANI, E.; MORNIROLI, A.; SORMANI, R. ; TONELLO, A.. **An experience report on using code smells detection tools.** In: 4TH INTERNATIONAL CONFERENCE ON SOFTWARE TESTING, VERIFICATION AND VALIDATION, ICST 2012, BERLIN, GERMANY, 21-25 MARCH, 2011, WORKSHOP PROCEEDINGS, p. 450–457, 2011.
- [29] FOWLER, M.. **Refactoring: Improving the Design of Existing Code.** Addison-Wesley, Boston, MA, USA, 1 edition edition, 1999.
- [30] GAMMA, E.; HELM, R.; JOHNSON, R. ; VLISSIDES, J.. **Design patterns: elements of reusable object-oriented software.** Pearson Education India, 1995.
- [31] GUPTA, M.; FERNANDEZ, J.. **How globally distributed software teams can improve their collaboration effectiveness?** In: GLOBAL SOFTWARE ENGINEERING (ICGSE), 2011 6TH IEEE INTERNATIONAL CONFERENCE ON, p. 185–189. IEEE, 2011.
- [32] HANNAY, J. E.; DYBÅ, T.; ARISHOLM, E. ; SJØBERG, D. I.. **The effectiveness of pair programming: A meta-analysis.** Information and Software Technology, 51(7):1110 – 1122, 2009. Special Section: Software Engineering for Secure Systems.
- [33] KHADKA, R.; BATLAJERY, B. V.; SAEIDI, A. M.; JANSEN, S. ; HAGE, J.. **How do professionals perceive legacy systems and software modernization?** In: PROCEEDINGS OF THE 36TH INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING, ICSE 2014, p. 36–47, Hyderabad, India, 2014. ACM.
- [34] KHOMH, F.; PENTA, M. D. ; GUEHENEUC, Y. G.. **An exploratory study of the impact of code smells on software change-proneness.** In: 2009 16TH WORKING CONFERENCE ON REVERSE ENGINEERING, p. 75–84, Oct 2009.

- [35] KRASNER, G.; POPE, S.. **A description of the Model-View-Controller user interface paradigm in the smalltalk-80 system.** Journal of Object Oriented Programming, 1(3):26–49, 1988.
- [36] LAMKANFI, A.; DEMEYER, S.; GIGER, E. ; GOETHALS, B.. **Predicting the severity of a reported bug.** In: MINING SOFTWARE REPOSITORIES (MSR), 2010 7TH IEEE WORKING CONFERENCE ON, p. 1–10. IEEE, 2010.
- [37] LANZA, M.; MARINESCU, R.. **Object-oriented metrics in practice: using software metrics to characterize, evaluate, and improve the design of object-oriented systems.** Springer Science & Business Media, 2007.
- [38] LAPKOVA, D.; ADAMEK, M.. **Statistical and mathematical classification of direct punch.** In: 2015 38TH INTERNATIONAL CONFERENCE ON TELECOMMUNICATIONS AND SIGNAL PROCESSING (TSP), p. 486–489, July 2015.
- [39] LEHMAN, M. M.. **Programs, life cycles, and laws of software evolution.** Proceedings of the IEEE, 68(9):1060–1076, 1980.
- [40] LEVENE, H.; OTHERS. **Robust tests for equality of variances.** Contributions to probability and statistics, 1:278–292, 1960.
- [41] LORENZ, M.; KIDD, J.. **Object-oriented software metrics: a practical guide.** Prentice-Hall, Inc., 1994.
- [42] LOZANO, A.; WERMELINGER, M.. **Assessing the effect of clones of changeability.** In: PROCEEDINGS OF THE 24TH IEEE INTERNATIONAL CONFERENCE ON SOFTWARE MAINTENANCE; BEIJING, CHINA, p. 227–236, 2008.
- [43] MACIA, I.; ARCOVERDE, R.; GARCIA, A.; CHAVEZ, C. ; VON STAA, A.. **On the Relevance of Code Anomalies for Identifying Architecture Degradation Symptoms.** p. 277–286. IEEE, mar 2012.
- [44] MACIA, I.; GARCIA, J.; POPESCU, D.; GARCIA, A.; MEDVIDOVIC, N. ; VON STAA, A.. **Are automatically-detected code anomalies relevant to architectural modularity?: An exploratory analysis of evolving systems.** In: PROCEEDINGS OF THE 11TH ANNUAL INTERNATIONAL CONFERENCE ON ASPECT-ORIENTED SOFTWARE DEVELOPMENT, AOSD '12, p. 167–178, Potsdam, Germany, 2012. ACM.

- [45] MANSOOR, U.; KESSENTINI, M.; MAXIM, B. R. ; DEB, K.. **Multi-objective code-smells detection using good and bad design examples.** *Software Quality Journal*, 25(2):529–552, 2017.
- [46] MARTICORENA, R.; LÓPEZ, C. ; CRESPO, Y.. **Extending a taxonomy of bad code smells with metrics.** In: 7TH INTERNATIONAL WORKSHOP ON OBJECT-ORIENTED REENGINEERING, WOOR '06, Nantes, France, 2006.
- [47] MCCONNELL, S.. **Code Complete, Second Edition.** Microsoft Press, Redmond, WA, USA, 2004.
- [48] MCINTOSH, S.; KAMEI, Y.; ADAMS, B. ; HASSAN, A. E.. **The impact of code review coverage and code review participation on software quality: A case study of the qt, vtk, and itk projects.** In: PROCEEDINGS OF THE 11TH WORKING CONFERENCE ON MINING SOFTWARE REPOSITORIES, p. 192–201, Hyderabad, India, 2014.
- [49] DE MELLO, R. M.; OLIVEIRA, R. ; GARCIA, A.. **Investigating the influence of human factors on the identification of code smells.** In: EMPIRICAL SOFTWARE ENGINEERING AND MEASUREMENT, p. 1–10, 2017.
- [50] MOHA, N.; GUEHENEUC, Y.; DUCHIEN, L. ; MEUR, A. L.. **Decor: A method for the specification and detection of code and design smells.** *IEEE Transaction on Software Engineering*, 36:20–36, 2010.
- [51] MURPHY-HILL, E.; PARNIN, C. ; BLACK, A. P.. **How we refactor, and how we know it.** *IEEE Transactions on Software Engineering*, 38(1):5–18, 2012.
- [52] MURPHY-HILL, E.; BLACK, A. P.. **Seven habits of a highly effective smell detector.** In: PROCEEDINGS OF THE 2008 INTERNATIONAL WORKSHOP ON RECOMMENDATION SYSTEMS FOR SOFTWARE ENGINEERING, RSSE '08, p. 36–40, Atlanta, Georgia, 2008. ACM.
- [53] MURPHY-HILL, E.; BLACK, A. P.. **An interactive ambient visualization for code smells.** In: PROCEEDINGS OF THE 5TH INTERNATIONAL SYMPOSIUM ON SOFTWARE VISUALIZATION, SOFTVIS '10, p. 5–14, Salt Lake City, Utah, USA, 2010. ACM.
- [54] MÄNTYLÄ, M.. **An experiment on subjective evolvability evaluation of object-oriented software: explaining factors and inter-rater agreement.** In: INTERNATIONAL SYMPOSIUM ON EMPIRICAL

- SOFTWARE ENGINEERING, ISESE, p. 287–296. IEEE Computer Society, 2005.
- [55] MÄNTYLÄ, M. V.; VANHANEN, J. ; LASSENIUS, C.. **Bad smells - humans as code critics.** In: SOFTWARE MAINTENANCE, 2004. PROCEEDINGS. 20TH IEEE INTERNATIONAL CONFERENCE ON, p. 399–408, 2004.
- [56] OIZUMI, W. N.; GARCIA, A. F.; COLANZI, T. E.; FERREIRA, M. ; STAA, A. V.. **On the relationship of code-anomaly agglomerations and architectural problems.** Journal of Software Engineering Research and Development, 3(1):11, 2015.
- [57] OIZUMI, W.; GARCIA, A.; DA SILVA SOUSA, L.; CAPEO, B. ; ZHAO, Y.. **Code anomalies flock together: Exploring code anomaly agglomerations for locating design problems.** In: PROCEEDINGS OF THE 38TH INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING, ICSE '16, p. 440–451, New York, NY, USA, 2016. ACM.
- [58] OLBRICH, S. M.; CRUZES, D. S. ; SJOBERG, D. I. K.. **Are all code smells harmful? A study of god classes and brain classes in the evolution of three open source systems.** In: PROCEEDINGS OF THE 26TH IEEE INTERNATIONAL CONFERENCE ON SOFTWARE MAINTENANCE, p. 1–10, Sept 2010.
- [59] OLBRICH, S.; CRUZES, D. S.; BASILI, V. ; ZAZWORKA, N.. **The evolution and impact of code smells: A case study of two open source systems.** In: PROCEEDINGS OF THE 2009 3RD INTERNATIONAL SYMPOSIUM ON EMPIRICAL SOFTWARE ENGINEERING AND MEASUREMENT, ESEM '09, p. 390–400, Washington, DC, USA, 2009. IEEE Computer Society.
- [60] OLIVEIRA, R.. **When more heads are better than one? understanding and improving collaborative identification of code smells.** In: ICSE '16 COMPANION, IN: DOCTORAL SYMPOSIUM, 2016, ICSE '16, p. 879–882, Austin, TX, USA, May. 2016.
- [61] OLIVEIRA, R.; GARCIA, A.; LUCENA, C. ; ALBUQUERQUE, D.. **A eficácia de pares na identificação de anomalias de código: Um experimento controlado.** In: 11TH WORKSHOP ON SOFTWARE MODULARITY (WMOD'14), 2014, p. 53–66, Maceió, Brazil, Oct. 2014.

- [62] OLIVEIRA, R.; ESTACIO, B.; GARCIA, A.; MARCZAK, S.; PRIKLADNICKI, R.; KALINOWSKI, M. ; LUCENA, C.. **Identifying code smells with collaborative practices: A controlled experiment**. In: 10TH BRAZILIAN SYMPOSIUM ON COMPONENTS, ARCHITECTURES, AND REUSE, SBCARS'16, p. 61–70, Maringá, Brazil, 2016b.
- [63] OLIVEIRA, R.; SOUSA, L.; MELLO, R.; VALENTIM, N.; LOPES, A.; CONTE, T.; GARCIA, A.; OLIVEIRA, E. ; LUCENA, C.. **Collaborative identification of code smells: A multi-case study**. In: SOFTWARE ENGINEERING IN PRACTICE, p. 33–42, Buenos Aires, Argentina, 2017.
- [64] OLIVEIRA, R.; MELLO, R.; GARCIA, A. ; LUCENA, C.. **Evaluating the effectiveness of pair programming on the identification of code smells: An empirical study**. 2017b.
- [65] PADILHA, J.; PEREIRA, J.; FIGUEIREDO, E.; ALMEIDA, J.; GARCIA, A.; SANT'ANNA, CLÁUDIO", E. M.; MYLOPOULOS, J.; QUIX, C.; ROLLAND, C.; MANOLOPOULOS, Y.; MOURATIDIS, H. ; HORKOFF, J.. **On the Effectiveness of Concern Metrics to Detect Code Smells: An Empirical Study**, p. 656–671. Springer International Publishing, Cham, 2014.
- [66] PAIVA, T.; DAMASCENO, A.; PADILHA, J.; FIGUEIREDO, E. ; SANT'ANNA, C.. **Experimental evaluation of code smell detection tools**. In: III Workshop DE Visualização, Evolução E Manutenção DE Software (VEM), p. 17–24, 2015.
- [67] PALOMBA, F.; OLIVETO, R. ; DE LUCIA, A.. **Investigating code smell co-occurrences using association rule learning: A replicated study**. In: MACHINE LEARNING TECHNIQUES FOR SOFTWARE QUALITY EVALUATION (MALTESQUE), IEEE WORKSHOP ON, p. 8–13. IEEE, 2017.
- [68] PARNAS, D. L.. **Software aging**. In: PROCEEDINGS OF THE 16TH INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING, ICSE 94, p. 279–287, Los Alamitos, CA, USA, 1994. IEEE Computer Society Press.
- [69] PELDSZUS, S.; KULCSÁR, G.; LOCHAU, M. ; SCHULZE, S.. **Continuous detection of design flaws in evolving object-oriented programs using incremental multi-pattern matching**. In: PROCEEDINGS OF THE 31ST IEEE/ACM INTERNATIONAL CONFERENCE ON AUTOMATED SOFTWARE ENGINEERING, ASE 2016, p. 578–589, Los Alamitos, CA, USA, 2016. IEEE Computer Society.

- [70] PHAPHOOM, N.; SUCCI, G.; VLASENKO, J.; DI BELLA, E.; FRONZA, I.; SILLITTI, A.. **Pair programming and software defects—a large, industrial case study**. IEEE Transactions on Software Engineering, 39:930–953, 2013.
- [71] PHILIPP, M.. **Qualitative content analysis**. FQS'00, 2000.
- [72] PIETRZAK, B.; WALTER, B.. **Leveraging code smell detection with inter-smell relations**. In: PROCEEDINGS OF THE 7TH INTERNATIONAL CONFERENCE ON EXTREME PROGRAMMING AND AGILE PROCESSES IN SOFTWARE ENGINEERING, XP'06, p. 75–84, Berlin, Heidelberg, 2006. Springer-Verlag.
- [73] RIGBY, P. C.; BIRD, C.. **Convergent contemporary software peer review practices**. In: PROCEEDINGS OF THE 2013 9TH JOINT MEETING ON FOUNDATIONS OF SOFTWARE ENGINEERING, ESEC/FSE 2013, p. 202–212, Saint Petersburg, Russia, 2013. ACM.
- [74] RIGBY, P.; CLEARY, B.; PAINCHAUD, F.; STOREY, M. A. ; GERMAN, D.. **Contemporary peer review in action: Lessons from open source development**. IEEE Software, 29(6):56–61, Nov 2012.
- [75] RUNESON, P.; HOST, M.; RAINER, A. ; REGNELL, B.. **Case Study Research in Software Engineering: Guidelines and Examples**. Wiley Publishing, 1st edition, 2012.
- [76] SIEGEL, S.; CASTELLAN, N. J. J.. **Nonparametric statistics for the behavioral sciences**. McGraw-Hill, New York, St. Louis, 1988.
- [77] SILVA, D.; TSANTALIS, N. ; VALENTE, M. T.. **Why we refactor? confessions of github contributors**. In: PROCEEDINGS OF THE 2016 24TH ACM SIGSOFT INTERNATIONAL SYMPOSIUM ON FOUNDATIONS OF SOFTWARE ENGINEERING, FSE 2016, p. 858–870, New York, NY, USA, 2016. ACM.
- [78] DAG I. K. SJOBERG, BENTE ANDA, E. A. T. D. M. J. A. K. E. F. K.; VOKÁC, M.. **Conducting realistic experiments in software engineering**. In: PROCEEDINGS OF THE 2002 INTERNATIONAL SYMPOSIUM ON EMPIRICAL SOFTWARE ENGINEERING, ISESE '02, p. 17–, Washington, DC, USA, 2002. IEEE Computer Society.
- [79] DAG I. K. SJØBERG, AIKO YAMASHITA, B. A. A. M.; DYBA, T.. **Quantifying the effect of code smells on maintenance effort**. volumen 39, p. 1144–1156, 2013.

- [80] STAMELOS, I. G.; SFETSOS, P.. **Agile software development quality assurance**. IGI Global, 2007.
- [81] STRAUSS, A.; CORBIN, J.. **Basics of Qualitative Research: Techniques and Procedures for Developing Grounded Theory**. SAGE Publications, 1998.
- [82] Succi, G.; Marchesi, M., editors. **Extreme Programming Examined**. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2001.
- [83] SUTHERLAND, A.; VENOLIA, G.. **Can peer code reviews be exploited for later information needs?** In: 2009 31ST INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING - COMPANION VOLUME, p. 259–262, May 2009.
- [84] TSANTALIS, N.; CHAIKALIS, T. ; CHATZIGEORGIOU, A.. **Jdeodorant: Identification and removal of type-checking bad smells**. In: SOFTWARE MAINTENANCE AND REENGINEERING, 2008. CSMR 2008. 12TH EUROPEAN CONFERENCE ON, p. 329–331. IEEE, 2008.
- [85] VANHANEN, J.; LASSENIUS, C. ; MÄNTYLÄ, M. V.. **Issues and tactics when adopting pair programming: A longitudinal case study**. In: INTERNATIONAL CONFERENCE SOFTWARE ENGINEERING ADVANCES, ICSEA, p. 70–70, Aug 2007.
- [86] VIDAL, S.; GUIMARAES, E.; OIZUMI, W.; GARCIA, A.; PACE, A. D. ; MARCOS, C.. **Identifying architectural problems through prioritization of code smells**. In: 2016 X BRAZILIAN SYMPOSIUM ON SOFTWARE COMPONENTS, ARCHITECTURES AND REUSE (SBCARS), p. 41–50, Sept 2016.
- [87] WALLE, T.; HANNAY, J. E.. **Personality and the nature of collaboration in pair programming**. In: 2009 3RD INTERNATIONAL SYMPOSIUM ON EMPIRICAL SOFTWARE ENGINEERING AND MEASUREMENT, p. 203–213, Oct 2009.
- [88] WHITEHEAD, J.. **Collaboration in software engineering: A roadmap**. In: 2007 FUTURE OF SOFTWARE ENGINEERING, FOSE '07, p. 214–225, Washington, DC, USA, 2007. IEEE Computer Society.
- [89] WOHLIN, C.; RUNESON, P.; HÖST, M.; OHLSSON, M. C.; REGNELL, B. ; WESSLÉN, A.. **Experimentation in Software Engineering: An Introduction**. Kluwer Academic Publishers, 2012.



- [90] YAMASHITA, A.; MOONEN, L.. Exploring the impact of inter-smell relations on software maintainability: An empirical study. In: PROCEEDINGS OF THE 2013 INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING, ICSE '13, p. 682–691, Piscataway, NJ, USA, 2013. IEEE Press.
- [91] YAMASHITA, A.; MOONEN, L.. Do developers care about code smells? an exploratory survey. In: 2013 20TH WORKING CONFERENCE ON REVERSE ENGINEERING (WCRE), p. 242–251, Oct 2013.
- [92] YAMASHITA, A.; MOONEN, L.. To what extent can maintenance problems be predicted by code smell detection? - an empirical study. Journal Information Software Technology, 55(12):2223–2242, Dec. 2013b.
- [93] KITCHENHAM, B. A.; PFLEEGER, S. L.; PICKARD, L. M.; JONES, P. W.; HOAGLIN, D. C.; EL EMAM, K. ; ROSENBERG, J.. Preliminary guidelines for empirical research in software engineering. IEEE Transactions on software engineering, 28(8):721–734, 2002.

## **A**

### **Appendix A - Consent Form**

This appendix presents the informed consent form used in the studies reported in Chapters 3, 4, and 5.

## Consent Term

PUC-RJ, through the OPUS Software Engineering research group, thanks you for your invaluable contribution to the advancement of the research in Software Engineering.

The goal of this research is to investigate the adoption of collaboration on the identification of code smells. For this purpose, participants are invited to perform a case study regarding the identification of code smells and to answer two questionnaires (before and after the activities). Two or more researchers will guide the participants during the task of identifying code smells.

We highlight that the goal of this study is not to evaluate the participants; instead, we evaluate the adoption of the collaboration on the identification of code smell. The records made during the study is strictly limited to research, ensuring that:

1. The participant's anonymity shall be preserved in all documents published in scientific forums (such as conferences, periodicals, journals and the like) or pedagogical (such as course presentation slides, and the like);
2. The participant may have access to copies of these documents after their publication.
3. The participant who feels constricted or uncomfortable during the completion of the experimental tasks may discontinue his participation. If so, researchers will discard the participant's data, in which will not be used for any purpose on the evaluation.
4. The participant has the right to express, on the date of the experiment, any additional restriction or condition that seems to apply to the items listed above (1, 2 and 3).
5. Researchers are allowed to use the collected data for any purpose, for instance academic or pedagogical purposes, as long as the usage of the data is in accordance with the conditions mentioned above.

**I declare that I agree with the above terms.**

**Subject name:** \_\_\_\_\_ **Date:** \_\_\_\_/\_\_\_\_/\_\_\_\_

**Researcher name:** \_\_\_\_\_

## **B**

### **Appendix B - Characterization Questionnaire of Chapter 3**

This appendix presents the subject characterization questionnaire used in the study reported in Chapter 3

## APPENDIX B

### Subject characterization questionnaire

Subject name: \_\_\_\_\_

Dear subject,

The overall goal this questionnaire is characterized your background. The answers that will be obtained through this questionnaire will allow us to identify some key characteristics about four knowledge areas: Programming language, Java language, Pair Programming, and Code Smells. Moreover, we also will determine your working experience. **All information collected in this questionnaire will be treated confidentially.**

#### General information

##### 1. Programming language

Regarding your experience with programming language, check the item that most apply to your degree of experience:

<input type="checkbox"/>	I never had any contact with a programming language
<input type="checkbox"/>	I had contact with one or more programming language in classes or instructional material
<input type="checkbox"/>	I had contact with programming language in the context of academic system
<input type="checkbox"/>	I had contact with programming language for at least one year in industrial systems

##### 2. Java Programming

Regarding your experience with Java programming, check the item that most apply to your degree of experience:

<input type="checkbox"/>	I never had contact with Java
<input type="checkbox"/>	I had contact with Java in classes or instructional material
<input type="checkbox"/>	I had contact with Java in the context of academic system
<input type="checkbox"/>	I had contact with Java for at least one year in industrial systems

### 3. Pair Programming

Pair Programming (PP) consists of two developers working together on the same programming activity - such as writing or editing the source code. Regarding your experience with pair programming, check the item that most apply to your degree of experience:

<input type="checkbox"/>	I never had contact with pair programming
<input type="checkbox"/>	I had contact with pair programming in classes or instructional material
<input type="checkbox"/>	I had contact with pair programming in the context of academic system
<input type="checkbox"/>	I had contact with pair programming for at least one year in industrial systems

### 4. Code Smells

Code smells are program structures that often indicate software maintainability problems. Regarding your experience with code smells, check the item that most apply to your degree of experience:

<input type="checkbox"/>	I never had contact with code smells
<input type="checkbox"/>	I had contact with code smells in classes or instructional material
<input type="checkbox"/>	I had contact with code smells in the context of academic system
<input type="checkbox"/>	I had contact with code smells for at least one year in industrial systems

**Date:** \_\_\_\_/\_\_\_\_/\_\_\_\_

**Researcher name:** \_\_\_\_\_

## **C**

### **Appendix C - Characterization Questionnaire of Chapter 4**

This appendix presents the subject characterization questionnaire used in the study reported in Chapter 4

## APPENDIX C

### Subject characterization questionnaire

Subject name: \_\_\_\_\_

Dear subject (a),

The overall goal of this questionnaire is to characterize your background. The answers that will be obtained through this questionnaire will allow us to identify some key characteristics about four knowledge areas: Programing language, Code Smells, Java language, and Pair Programming. Moreover, we also will determine your working experience. **All information collected in this questionnaire will be treated confidentially.**

#### General information

##### 1. Programming language

Regarding your experience with programming language, check the items that most apply to your degree of experience:

<input type="checkbox"/>	I never had any contact with a programming language
<input type="checkbox"/>	I had contact with one or more programming languages in classes or instructional material
<input type="checkbox"/>	I had contact with programming language in the context of academic system
<input type="checkbox"/>	I had contact with programming language in the context of industrial software systems

##### 2. Java Programming

Regarding your experience with Java programming, check the item that most apply to your degree of experience:

<input type="checkbox"/>	I never had contact with Java
<input type="checkbox"/>	I had contact with Java in classes or instructional material
<input type="checkbox"/>	I had contact with Java in the context of academic system
<input type="checkbox"/>	I had contact with Java in the context of industrial software systems



### 3. Pair Programming

Pair Programming (PP) consists of two developers working together on the same programming activity - such as writing or editing the source code. Regarding your experience with pair programming, check the items that most apply to your degree of experience:

<input type="checkbox"/>	I never had contact with pair programming
<input type="checkbox"/>	I had contact with pair programming in classes or instructional material
<input type="checkbox"/>	I had contact with pair programming in the context of academic system
<input type="checkbox"/>	I had contact with pair programming in the context of industrial software systems

### 4. Code Smells

Code smells are program structures that often indicate software maintainability problems. Regarding your experience with code smells, check the item that most apply to your degree of experience:

<input type="checkbox"/>	I never had contact with code smells
<input type="checkbox"/>	I had contact with code smells in classes or instructional material
<input type="checkbox"/>	I had contact with code smells in the context of academic system
<input type="checkbox"/>	I had contact with code smells in the context of industrial software systems

**Date:** \_\_\_\_/\_\_\_\_/\_\_\_\_

**Researcher name:** \_\_\_\_\_

## **D**

### **Appendix D - Characterization Questionnaire of Chapter 5**

This appendix presents the subject characterization questionnaire used in the study reported in Chapter 5

## APPENDIX D

### Subject characterization questionnaire

Subject name: \_\_\_\_\_

Dear subject (a),

The overall goal of this questionnaire is to characterize your working experience. The answers that will be obtained through this questionnaire will allow us to identify some key characteristics about three knowledge areas: Java language, Software development, and Code review. **All information collected in this questionnaire will be treated confidentially.**

#### General information

**1. Select your current degree:**

<input type="checkbox"/>	I don't have a formal education in computer science
<input type="checkbox"/>	Technologist
<input type="checkbox"/>	BSc
<input type="checkbox"/>	Master
<input type="checkbox"/>	PhD

**2. Experience with software development (in years):**

\_\_\_\_\_

**3. Experience with the Java programming language (in years):**

\_\_\_\_\_

**4. How many Java softwares have you worked with peer review?**

\_\_\_\_\_

**5. Which are the names of Java systems that you are or were involved in the current organization?**

---

**6. Which are your current positions in the organization?**

	software developer
	Project manager
	Technical leader
	Consultant
	Other:

**Date:** \_\_\_\_/\_\_\_\_/\_\_\_\_

**Researcher name:** \_\_\_\_\_

## **E**

### **Appendix E - Code Smell Report Questionnaire**

This appendix presents the Code smell report questionnaire used in the studies reported in Chapters 3, 4, and 5.

## APPENDIX E

### Code smell report questionnaire

Subject name: \_\_\_\_\_

System name: \_\_\_\_\_

Dear subject (a),

The overall goal of this questionnaire is to collect the smell types which you identified in the system. All information collected in this questionnaire will be treated confidentially.

Sample answer:

1 – Code element **X** has smell type **Y**. Because ...  
2 - ...  
3 - ...  
...

## **F**

### **Appendix F - Follow-Up Questionnaire of Chapter 3**

This appendix presents the follow-up questionnaire used in the study reported in Chapter 3

## APPENDIX F

### Follow-up questionnaire

Subject name: \_\_\_\_\_

Dear subject (a),

The overall goal of this questionnaire is to acquire information about on the identification task of code smells. **All information collected in this questionnaire will be treated confidentially.**

#### General information

1. In your opinion, which code smell types were more difficult to identify? Why?

\_\_\_\_\_

2. Regarding the previous question, which steps you followed to identify it?

\_\_\_\_\_

3. In your opinion, which code smell types were easier to identify? Why?

\_\_\_\_\_

4. In your opinion, which code smell types did the collaboration help you to identify?

\_\_\_\_\_



5. In your opinion, what were the benefits of working collaboratively (Pair programming) in the identification task of code smells?

---

6. In your opinion, what were the disadvantages of working collaboratively (Pair programming) in the identification of code smells?

---

Date: \_\_/\_\_/\_\_

## **G**

### **Appendix G - Follow-Up Questionnaire of Chapter 4**

This appendix presents the follow-up questionnaire used in the study reported in Chapter 4

## APPENDIX G

### Follow-up questionnaire

Subject name: \_\_\_\_\_

Dear subject (a),

The overall goal of this questionnaire is to acquire information about on the identification task of code smells. **All information collected in this questionnaire will be treated confidentially.**

#### General information

1. In your opinion, which code smell types were more difficult to identify? Why?

\_\_\_\_\_

2. Regarding the previous question, which steps you followed to identify it?

\_\_\_\_\_

3. In your opinion, which code smell types were easier to identify? Why?

\_\_\_\_\_

4. In your opinion, which code smell types did the collaboration help you to identify?

\_\_\_\_\_

5. In your opinion, what were the benefits of working collaboratively (in pairs and in groups) in the identification of code smells?
- 

6. In your opinion, what were the disadvantages of working collaboratively (in pairs and in groups) in the identification task of code smells?
- 

Date: \_\_/\_\_/\_\_

## **H**

### **Appendix H - Follow-Up Questionnaire of Chapter 5**

This appendix presents the follow-up questionnaire used in the study reported in Chapter 5

## APPENDIX H

### Follow-up questionnaire

Subject name: \_\_\_\_\_

Dear subject (a),

The overall goal of this questionnaire is to acquire information about on the identification task of code smells. **All information collected in this questionnaire will be treated confidentially.**

#### General information

1. In your opinion, which code smell types were more difficult to identify? Why?

\_\_\_\_\_

2. Regarding the previous question, which steps you followed to identify it?

\_\_\_\_\_

3. In your opinion, which code smell types were easier to identify? Why?

\_\_\_\_\_

4. Regarding the previous question, which steps you followed to identify it?

\_\_\_\_\_

5. Which information do you believe that would help you to identify code smells but it was not provided?

\_\_\_\_\_

6. In your opinion, which code smell types did the collaboration help you to identify?

---

7. In your opinion, what were the benefits of working collaboratively in the identification of code smells?

---

8. In your opinion, what were the disadvantages of working collaboratively in the identification task of code smells?

---

Date: \_\_/\_\_/\_\_

# I

## **Appendix I - Organization Characterization**

This appendix presents the organization characterization questionnaire used in the study reported in Chapter 5



## APPENDIX I

### Organization characterization questionnaire

Employee's name: \_\_\_\_\_

Dear (a),

The overall goal of this form is to characterize the company mentioned below. The answers that will be obtained through this form will allow us to identify some key characteristics about the organization. **All information collected in this questionnaire will be treated confidentially.**

#### General information

1. Organization's name:

\_\_\_\_\_

2. Organization's type:

<input type="checkbox"/>	Private
<input type="checkbox"/>	Governmental
<input type="checkbox"/>	Other:

3. How many developers work on the organization in the software development area?

\_\_\_\_\_

4. How many developers does your software team have?

\_\_\_\_\_

#### Developed Products and Services

5. Does the organization operate in foreign markets?

\_\_\_\_\_

**6. Which are the main categories of products developed by the organization? (Choose one or more)**

	Software to support the organization
	Off-the-shelf-software
	Custom service
	Other(s):

**7. Which are the domains of software systems developed by the organization (Choose one or more)**

	Human resources management
	Data services
	Banking system software
	Commercial automation systems
	E-business software
	IDE
	Database management
	Game
	Web pages
	Security systems
	Other:

### **Methodologies and process applied in the organization**

**8. What are the development processes adopted by organization?**

---

**9. What are the programming languages adopted by the organization?**

---

**10. What are the agile development methodologies adopted by the organization?**

---

**11. Does the organization perform code review?**

---


**Date:** \_\_\_\_/\_\_\_\_/\_\_\_\_

## **J**

### **Appendix J - Support Material for Training**

This appendix presents the support material for training used in the studies reported in Chapters 3, 4, and 5.

PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO DE JANEIRO

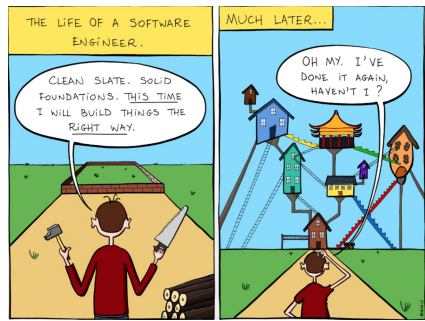


## IDENTIFYING CODE SMELLS

LES | DI | PUC-Rio - Brazil

## Development Process


### Warming-up



2

## Code Smells

- ◆ Code smell is an implementation structure that indicates a deeper problem in the system
- ◆ They have been used as indicators of design problems in the system
- ◆ They are symptoms of bad design or bad implementation choices



This Stinks

<sup>1</sup>Fowler, M. et al. Refactoring: Improving the Design of Existing Code. Addison-Wesley Professional, 1999

3

## Challenges of the Code Smell Identification

- ◆ There are more than 20 catalogued code smells<sup>1</sup>
- ◆ Smell identification often requires subjective analysis made by software developers<sup>2</sup>
- ◆ Developers might miss some code smells

<sup>1</sup>Fowler, M. et al. Refactoring: Improving the Design of Existing Code. Addison-Wesley Professional, 1999

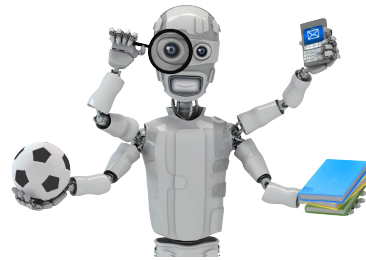
<sup>2</sup>Bernhart, M. and Grechening, T. On the Understanding of Programs with Continuous Code Reviews. In. Program Comprehension, 2013.

4



## Examples of Code Smells

5

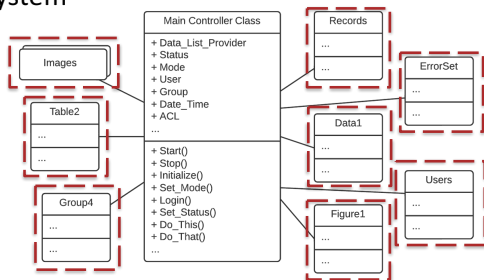


## God Class

6

## God Class

- ◆ A class that centralizes the intelligence of the system

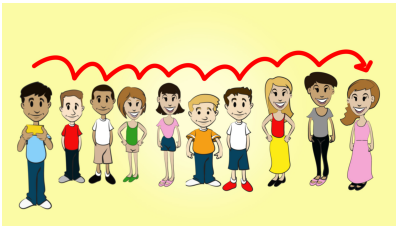


7

## God Class – Symptoms

- ◆ Class is complex for reuse and test
  - ◆ It is hard for understanding
  - ◆ It is hard for testing
  - ◆ It is inefficient for reuse

8




**Message Chains**

9

**Message Chains**

- ◆ Long sequence of method calls to get a data

**A.b().c().d().e()**




10


**Message Chains – Consequences**

- ◆ It increases the dependency between classes of a chain

**A.b().c()**



11

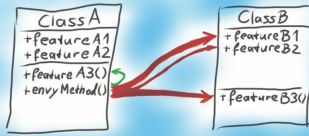


**Feature Envy**

12

## Feature Envy

- ◆ A method accesses the data of another object more than its own data



13

## Feature Envy - Symptoms

- ◆ Method needs data from one or more classes to perform an action
- ◆ Method almost never accesses the attributes and methods from its own class

14

## Feature Envy - Example

```
public class Phone {
    private final String unformattedNumber;
    public Phone(String unformattedNumber) {
        this.unformattedNumber = unformattedNumber;
    }
    public String getAreaCode() {
        return unformattedNumber.substring(0,3);
    }
    public String getPrefix() {
        return unformattedNumber.substring(3,6);
    }
    public String getNumber() {
        return unformattedNumber.substring(6,10);
    }
}

public class Customer {
    private Phone mobilePhone;
    public String getMobilePhoneNumber() {
        return "(" +
            mobilePhone.getAreaCode() + ") " +
            mobilePhone.getPrefix() + "-" +
            mobilePhone.getNumber();
    }
}
```

(a)

```
public class Phone {
    private final String unformattedNumber;
    public Phone(String unformattedNumber) {
        this.unformattedNumber = unformattedNumber;
    }
    private String getAreaCode() {
        return unformattedNumber.substring(0,3);
    }
    private String getPrefix() {
        return unformattedNumber.substring(3,6);
    }
    private String getNumber() {
        return unformattedNumber.substring(6,10);
    }
    public String toFormattedString() {
        return "(" + getAreaCode() + ") " +
            getPrefix() + "-" + getNumber();
    }
}

public class Customer {
    private Phone mobilePhone;
    public String getMobilePhoneNumber() {
        return mobilePhone.toFormattedString();
    }
}
```

(b)

15



## More Examples

16

### Long Method

Definition and Example

- Method that contains too many lines of code and is overload of functionalities

```
void megazord (double quantia){
    // procedure1
    ....
    // procedure30
}
```

17

### Lazy Class

Definition and Example

- Class that does too little

```
public class Letter {
    private final String content;

    public Letter(String content) {
        this.content = content;
    }

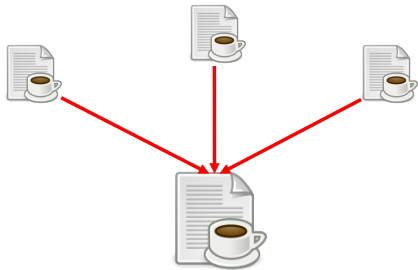
    public String getContent() {
        return content;
    }
}
```

18

### Divergent Change

Definition and Example

- Class that is commonly changed in different ways for different reasons

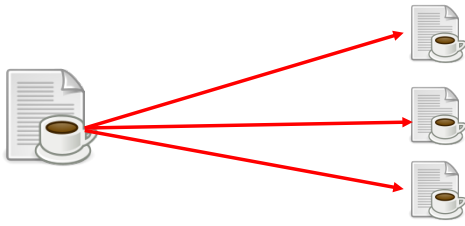


19

### Shotgun Surgery

Definition and Example

- Class that triggers many small changes to many different classes when it is modified



20



## Refused Bequest

### Definition and Example

- Subclass that rejects or invalidates the methods supported from its superclass


```
public abstract class AbstractCollection{
    public abstract void add(Object element);
}

public class Map extends AbstractCollection{
    public void add(Object element){
        //
    }
}
```

21


## Other Code Smells

- Data Class**
  - Classes that have only fields as well as getting and setting methods, and nothing else
- Data Clumps**
  - Clusters of data that are often seen together as class members or in method signatures but are not grouped in a class
- Spaghetti Code**
  - A class without structure that declares long methods without parameters
- Duplicated Code**
  - The same code structure appears in more than one place



22

PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO DE JANEIRO



## PAIR PROGRAMMING

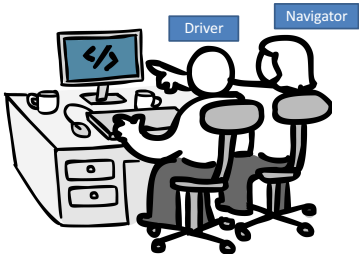
LES | DI | PUC-Rio - Brazil

## What is Pair Programing?

- Two developers working on the same task (as a team)
- Both have the same target
- Both have different expertise
- One executes the task
- The other looks for errors, reviews the implementation, and proposes error fixes

24

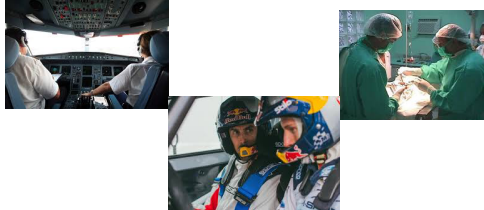
## Pairing Settings



25

## Is it worthwhile?

- ◆ Two developers will do the work of one
- ◆ Less work will get done
- ◆ Why would I put two people on a job that just one can do?

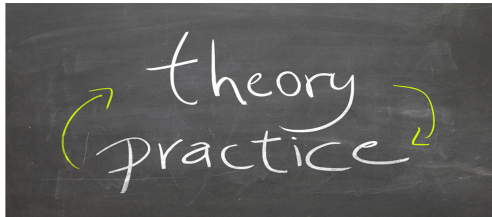


26

## How does it Help?

- ▶ Continuous review
- ▶ Less defects
- ▶ Defects caught early
- ▶ Improvement in the quality of the design
- ▶ Better problem solving
- ▶ Pair-pressure ensure timely delivery
- ▶ Better induction of new team members
- ◆ Saved effort on intra-team documentation
- ◆ Improved satisfaction
- ◆ Better team building and communication

27



## Practical Activity

28

Experiment Activity

◆ General Goal

◆ Identify code smells in a system

◆ Procedure:

Code Smell List

Practical Activity

Identified Code Smells

29

Questions

30