Surveying Developer Knowledge and Interest in Code Smells through Online Freelance Marketplaces

Aiko Yamashita
Simula Research Laboratory
Lysaker, Norway
Email: aiko@simula.no

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

***Abstract*—This paper discusses the use of *freelance marketplaces* to conduct a survey amongst professional developer's about specific software engineering phenomena, in our case their knowledge and interest in code smells and their detection/removal. We present the context and motivation of our research, and the idea of using freelance marketplaces for conducting studies involving software professionals. Next, we describe the design of the survey and the specifics on the selected freelance marketplace (i.e., Freelancer.com). Finally, we discuss why freelance markets constitute a feasible and advantageous approach for conducting user evaluations that involve large numbers of software professionals, and what challenges such an approach may entail.**

***Index Terms*—survey; user evaluation; developer knowledge**

I. Introduction and Motivation

The presence of code smells indicates that there are issues with code quality, such as understandability and changeability, which can lead to a variety of maintenance problems, including the introduction of faults [1]. In the last decade, code smells have become an established concept for patterns or aspects of software design that may cause problems for further development and maintenance of these systems [2].

Since code smells are associated to specific set of refactoring strategies to eliminate them, code smell analysis allows people to integrate both *assessment* and *improvement* into the software evolution process itself. Van Emden and Moonen [3] provided the first formalization of code smell detection and developed an automated code smell detection tool for Java. Other approaches for code smell detection can be found in [4–13]. Automated code smell detection has been implemented in commercial tools such as Together[1], Analyst4J[2], Stan4J[3], InCode[4], NDepend[5], and CppDepend[6], and in free tools like JDeodorant[7], and OClint[8].

Even though code smell detection and removal has been well-researched over the last decade, it remains open to debate whether or not code smells should be considered meaningful conceptualizations of code quality issues from the developer's perspective. For example, the authors on a recent study on the lifespan of code smells in seven open source systems conclude that developers are aware but not concerned by the existence of code smells [14]. In another study, Yamashita and Moonen investigate how code smells relate to maintainability characteristics considered important by professional developers [15]. They found that although some concepts covered by code smells reflect maintainability properties important to developers, a considerable percentage of maintainability properties were unrelated to code smells.

So, the question remains if code smells are really important to developers? If they are not, is this due to the lack of *relevance of the underlying concepts* (e.g., as investigated in [15]), a lack of *awareness about code smells* on the developer's side, or due to the lack of *appropriate tools* for code smell analysis and/or removal? If tool support is lacking, what features would best support developers' needs? Thus, to align and direct research efforts to address actual needs and problems of professional developers, we need to better understand their level of knowledge and interest in code smells.

A good starting point to investigate these aspects would be an exploratory, descriptive survey. However, to conduct such a study, one needs a large enough, and representative sample of professional software engineers, which is generally difficult to attain. In this paper, we suggest the use of online freelance marketplaces for obtaining such a sample. The remainder of this paper is as follows: First, we briefly discuss the challenge in Software Engineering (SE) research of accessing representative samples in controlled studies. Second, we introduce outsourcing on freelance marketplaces as an approach to overcome the "sampling" challenge. Third, we describe the design of our survey study and the specifics on the particular freelance marketplace that we selected (i.e., Freelancer.com). Finally, we discuss why the use of freelance markets constitutes an advantageous approach for conducting user evaluations involving large numbers of software professionals, and what challenges such an approach may entail.

II. The Challenge of Representative Samples in SE

As mentioned above, our goal was to conduct a survey to investigate developer's insights and interest in code smells. A survey is defined as: "A system for collecting information from or about people to describe, compare or explain their knowledge, attitudes and behavior" [16].

An important challenge in Software Engineering (SE) research is to generalize from the specific subjects, technology, tasks and systems of experiments to industrial contexts [17]. Controlled studies that are performed on small systems or in unrealistic settings suffer from threats to their external validity.

[1] http://www.borland.com/us/products/together [2] http://www.codeswat.com
[3] http://wwww.stan4j.com [4] http://www.intooitus.com/products/incode
[5] http://www.ndepend.com [6] http://www.cppdepend.com
[7] http://www.jdeodorant.com [8] http://oclint.org

The same problem applies to the selection of participants or subjects in user evaluations and surveys, as they need to be representative of the population that we want to study.

Consequently, it is important for the purpose of our study to get access to software professionals rather than students or academic personnel. However, getting access to professional software engineers for conducting studies is often challenging, both in terms of establishing the contact with software companies, and in terms of convincing them to invest their time participating in such studies. Advocates of *Action Research* [18] suggest that this methodology enables the conduction of empirical studies in industry-relevant settings. However, action research is only one type of research methodology, and only suited for addressing certain types of research questions; conducting a wide exploratory survey is not one of them.

III. Online Freelance Marketplaces

Sjøberg et al., [17] argue that universities or research institutions should allocate resources for enabling realistic software engineering experiments in the same way as they allocate resources for acquiring software tools or equipment necessary for research. However, even if we have enough resources to "hire" programmers to participate, whenever such studies require a large population, it may still be challenging to reach out to enough software professionals to get a representative sample. We propose the use of *online freelance marketplaces* (also know as "outsourcing markets") as a feasible approach for reaching out to large numbers of software professionals for conducting such surveys or experiments.

Bacon et al., [19] define online freelance marketplaces as *"platforms that connect individuals, small-business owners, and even Fortune 500 companies with freelance technology specialists to satisfy their technological needs"*. They are supported by websites that provide detailed information on the freelancers' history and qualifications, and even provide tools for monitoring the progress of the outsourced software projects. Project owners post details of the project they offer and freelancers interested in taking the assignment will reply with their offers. The marketplace typically assures that buyers get the service they ordered and freelancers get payed when they complete the assignment. Examples of (IT) marketplaces discussed in [19] include *RentACoder* and *TopCoder*.

Freelance marketplaces are an example of *crowdsourcing* [20], essentially the assignment or distribution of tasks to a large group of people (often a community) instead of hiring or outsourcing to specific individuals. Crowdsourcing is based on announcing an open call to carry out a task, often involving a payments, prizes or other types of rewards [21]. One particularity of crowdsourcing is that participants in general have minimal or no interaction with each other (i.e., workers are independent and typically cannot see each others' work). This makes this environment ideal for conducting studies that demand a degree of control to avoid potential biases on the participant's responses. Alonso et al., propose the use of crowdsourcing for conducting relevance evaluation of information retrieval systems [21]. Along the same lines, we argue that online freelance marketplaces offer a platform that is well-suited to reach out to large numbers of software professionals, and at the same time provides a reward system to motivate participation in the proposed study.

IV. Design of Survey

A brainstorming session was organized to define goals and scope of the survey. The main goal is exploring the level of insight (i.e., awareness, knowledge) developers have on code smells, and determining if and why they are interested in code smell-related concepts and tools. A secondary goal is to explore how code smells (concept or tools) are currently used within industry, and how can they potentially be used within industry. We defined a set of background information to collect to characterize developer profiles. This information includes predominant working roles, programming language expertise, familiarity with programming paradigms, and working experience (hours and LOC). The format of the survey is an *exploratory, descriptive survey* [16], consisting of a combination of closed and open questions. Open questions are an important component of *qualitative surveys*, which are especially suited in cases where previous experiences or literature are insufficient to guide the design of closed questions [16]. To investigate the concern about, and perceived criticality of code smells, a 5-point Likert-Scale is used. Appendix A gives an overview of the questions for the survey. As one of the reviewers remarked, although we did "cross-examine" the questions with a colleague researcher, we overlooked the option of doing a real pilot study, which could have helped with ensuring the usefulness of the questions and exposing questions that were open to misinterpretation.

The marketplace that was selected to conduct the survey was *Freelancer.com*. This is a revised version of the *RentACoder* platform discussed in [19] after a merging and rebranding process. Projects on Freelancer.com are protected by an escrow service and through arbitration. "Buyers" can post calls for new projects on the site and "sellers" (freelancers) can ask questions, provide proposals for solutions, and submit bids on the projects. After the bidding period ends, the buyer awards the project to one or more sellers of choice and at the same time places the funds for payment into into escrow as a payment guarantee. After the work is completed, the buyer releases funds from the escrow account to the seller. If any of the parties fail to deliver the project or release the escrow, an arbiter will step in and resolve the conflict. Instead of bidding, another possible arrangement in Freelancer.com is "Pay for Time" where the buyer pays a freelancer for the time spent on a task rather than for a concrete outcome. To support the selection process, buyers and sellers may rate each other after work is completed. This "customer" rating system is different from TopCoder, where most of the reputation information if computed from measurable performance metrics such as the percentage of tests passed by the code, etc. We selected Freelancer.com because this marketplace offered the "Pay for Time" option that we considered ideal for rewarding tasks such as completing a questionnaire that comprise a relatively short

period of time, and because colleagues at Simula Research Laboratory had good experiences with using this platform.

V. Opportunities and Challenges of Conducting Studies in Freelance Marketplaces

The use of freelance marketplaces to conduct studies on software professionals has several advantages worth considering:

Flexibility: The survey was conducted as part of a larger online estimation experiment conducted by Simula, which took approximately two weeks to complete. This implies that in many cases, you could collect different kinds of data from the same population, in a fast, and efficient way. As Alonso et al., [21] points out, the low cost makes it possible to apply different methods for collecting different kinds of data.

Access to wide populations: 68 developers from 29 countries answered our survey. Had a similar population been contacted via more traditional methods, it would have been rather difficult to reach such an international, large number of participants within few weeks. The fact that one has access to large populations also allows to eliminate noise and support internal validation.

Relatively Low Cost: Developers charged from 15-20USD per hour for completing the online experiment and answering our survey questions. Costs may significantly increase if a large population is required, but for user evaluations, the costs remain relatively low if the amount of data that can be extracted is taken into consideration. Also, depending of the type of study and tasks required by the developers, the option of *contests* with a limited number of *prizes* could be an alternative option.

Along with the advantages that this approach brings, some challenges are yet to be tackled. One of the biggest challenges is the ***uncertainty with respect to background and skills*** of the participants. Verifying aspects such as competence, education and experience may be problematic. In our study we relied on self-assessment/self-reporting of skills. This could be replaced by running skill assessments on the participants at higher costs for participation. However, just as much aspects of the background of the participants may remain uncertain, and can pose a threat to internal validity. This might be particularly challenging in marketplaces such as Freelance.com, where rankings are based on subjective assessment of satisfaction from previous buyers. Some other marketplaces use more objective measurements of competence and skill, such as number of successful test cases of delivered code, participation in competitions and passing certification exams, etc.

A related challenge is ***ambiguity about representation***; many freelancers are registered on the marketplace as organizations, although this includes a large number of one-person companies. This means that if we want to conduct a study investigating aspects pertaining to organizations, we need to be careful that the selected organizations are of appropriate structure and size. This ambiguity can also pose a reverse threat: some large organizations are represented by a single intermediary, thereby appearing as a single developer. This makes it hard to establish a user profile based on previous ratings or projects. Moreover, it remains an open question who will actually participate in your survey: their non-English-speaking top developer that achieved their high rating, or their near-native intermediary who has never written a line of code.

Finally, it is not clear to what extend participants to these online freelance markets are ***representative of professional developers*** in general. The characteristics of these marketplaces could attract a certain type of developer or promote a certain type of behavior that differs from other software engineering contexts and this needs to be further investigated.

References

[1] M. Fowler, *Refactoring: Improving the Design of Existing Code*. Addison-Wesley, 1999.

[2] M. Lanza and R. Marinescu, *Object-Oriented Metrics in Practice*. Springer, 2005.

[3] E. Van Emden and L. Moonen, "Java quality assurance by detecting code smells," in *Working Conf. Reverse Eng. (WCRE)*, 2001, pp. 97–106.

[4] R. Marinescu and D. Ratiu, "Quantifying the quality of object-oriented design: the factor-strategy model," in *Working Conf. Reverse Eng. (WCRE)*. IEEE, 2004, pp. 192–201.

[5] R. Marinescu, "Measurement and quality in object-oriented design," in *IEEE Int'l Conf. Softw. Maintenance (ICSM)*, 2005, pp. 701–704.

[6] N. Moha, Y.-G. Guéhéneuc, and P. Leduc, "Automatic generation of detection algorithms for design defects," in *IEEE/ACM Int'l Conf. on Automated Softw. Eng.*, 2006, pp. 297–300.

[7] N. Moha, "Detection and correction of design defects in object-oriented designs," in *ACM SIGPLAN Conf. Object-oriented programming, systems, languages, and applications (OOPSLA)*, 2007, pp. 949–950.

[8] N. Moha, Y.-G. Guéhéneuc, A.-F. Le Meur, and L. Duchien, "A domain analysis to specify design defects and generate detection algorithms," in *Fundamental Approaches to Softw. Eng.*, 2008, pp. 276–291.

[9] A. A. Rao and K. N. Reddy, "Detecting bad smells in object oriented design using design change propagation probability matrix," in *Int'l Multiconf. of Eng. and Computer Scientists*, 2008, pp. 1001–1007.

[10] E. H. Alikacem and H. A. Sahraoui, "A Metric Extraction Framework Based on a High-Level Description Language," in *IEEE Int'l Conf. Source Code Analysis and Manipulation (SCAM)*, 2009, pp. 159–167.

[11] F. Khomh, M. Di Penta, and Y.-G. Guéhéneuc, "An Exploratory Study of the Impact of Code Smells on Software Change-proneness," in *Working Conf. Reverse Eng. (WCRE)*. IEEE, 2009, pp. 75–84.

[12] N. Moha, Y.-G. Guéhéneuc, L. Duchien, and A.-F. Le Meur, "DECOR: A Method for the Specification and Detection of Code and Design Smells," *IEEE Transactions on Software Engineering*, vol. 36, no. 1, pp. 20–36, 2010.

[13] N. Moha, Y.-G. Guéhéneuc, A.-F. Le Meur, L. Duchien, and A. Tiberghien, "From a domain analysis to the specification and detection of code and design smells," *Formal Aspects of Computing*, vol. 22, no. 3, pp. 345–361, 2010.

[14] R. Peters and A. Zaidman, "Evaluating the Lifespan of Code Smells using Software Repository Mining," *European Conf. Softw. Maint. and Reeng.*, pp. 411–416, 2012.

[15] A. Yamashita and L. Moonen, "Do code smells reflect important maintainability aspects?" in *IEEE Int'l Conf. Softw. Maintenance (ICSM)*. IEEE, 2012, pp. 306–315.

[16] A. Fink, *The Survey Handbook*, 2nd ed. Thousand Oaks, California: SAGE Publications, Inc., 2003.

[17] D. I. K. Sjøberg, B. Anda, E. Arisholm, T. Dyba, M. Jorgensen, A. Karahasanovic, E. F. Koren, and M. Vokac, "Conducting realistic experiments in software engineering," in *Int'l Symposium on Empirical Softw. Eng.*, 2002, pp. 17–26.

[18] D. E. Avison, F. Lau, M. D. Myers, and P. A. Nielsen, "Action research," *Communications of the ACM*, vol. 42, no. 1, pp. 94–97, 1999.

[19] D. F. Bacon, Y. Chen, D. Parkes, and M. Rao, "A market-based approach to software evolution," in *ACM SIGPLAN Conf. Object-oriented programming, systems, languages, and applications (OOPSLA)*. ACM, 2009, p. 973.

[20] J. Howe, "The Rise of Crowdsourcing," *Wired*, 2006.

[21] O. Alonso, D. E. Rose, and B. Stewart, "Crowdsourcing for relevance evaluation," *ACM SIGIR Forum*, vol. 42, no. 2, p. 9, 2008.

APPENDIX A: SURVEY QUESTIONS

Section I: Background

1. What is your predominant role within your organization?
   [ ] Developer
   [ ] Team Lead
   [ ] Tester
   [ ] Architect
   [ ] QA Manager
   [ ] Project Manager
   [ ] Self-employed

2. What is your level of skill in the following languages? (1=novice, 5=expert):

Language	Level
Java	
C	
C++	
C#	
Python	
Javascript	
VisualBasic	
Other (mention)	

3. What is your level of experience (LOC and months) in the following languages?

Language	LOC	Months
Java		
C		
C++		
C#		
Python		
Javascript		
VisualBasic		
Other (mention)		

4. Rank the following programming paradigms according to how familiar you are with each? (1=least familiar, 5=most familiar)

Paradigm	Familiarity
Functional	
Imperative	
Object Oriented	

Section II: Code Smells

5. How familiar are you with code smells or design anti-patterns? (choose one)
   [ ] I have never heard of them.
   [ ] I have heard about them in blogs or discussions but I am not so sure what they are.
   [ ] I have a general understanding, but do not use these concepts.
   [ ] I have a good understanding, and use these concepts sometimes.
   [ ] I have a strong understanding, and use these concepts frequently.

6. What are the sources from which you learn on code smells? (multiple choices)
   [ ] Blogs
   [ ] Discussion Forums
   [ ] Guru's websites
   [ ] Books
   [ ] Research Papers
   [ ] Tool vendors' websites

7. How concerned are you with the presence of code smells or anti-patterns in your code? (1=not concerned, 5=very concerned) Why?

8. Are there specific code smells / anti-patterns that you are concerned about? Please list them in order of their perceived importance.

9. Rank the situations where do you think code smell analysis/tools can be helpful (1=not helpful, 5=essential)

Situation	Level
Refactoring guidance (to find out where to refactor)	
Quality assessment (e.g., certification processes)	
Bug prediction (to identify areas of the code likely to have more defects)	
Effort prediction (to identify areas of the code likely to consume more time)	
Code inspection (to prioritize areas of the code to inspect)	
Others (mention)	

10. Have you used tools for detecting/analyzing code smells? Which ones?

11. Did you find the tools useful? Why/why not?

12. What features would you like in a tool for supporting detection or analysis of code smells? (list the most important ones first).

No.	Feature

13. Do you remove code smells "on the fly" or you plan and allocate time to "cleanup your code"? (choose one)
   [ ] On the fly
   [ ] Plan
   [ ] Combination

Section III: Removal of code-smells

14. How often do you refactor to remove code smells? (choose one)
   [ ] Never
   [ ] Almost never
   [ ] Sometimes, when is absolutely essential
   [ ] In a regular basis
   [ ] Constitutes an important part of the process (refactoring is included as a formal activity within the projects, f. ex. testing)

15. Can you characterize how much (seldom/regularly/often) of the refactoring is manual, tool assisted or combined?

Method	Frequency
Manual	
Tool assisted	
Combined	

16. Can you estimate how much (seldom/regularly/often) of the refactoring done is of low (renaming methods), of medium (relocating classes, extracting methods) or high (modify large segments of the code, replace solutions with the usage of patterns, etc.) complexity?

Refactoring complexity	Frequency
Low	
Medium	
High	

17. Would you like to know more about code smells / anti-patterns or refactoring? Why?