# Can you tell me if it smells? A study on how developers discuss code smells and anti-patterns in Stack Overflow

Amjed Tahir
School of Eng. and Advanced Tech.
Massey University
Palmerston North, New Zealand
a.tahir@massey.ac.nz

Aiko Yamashita
Department of Computer Science
Oslo Metropolitan University
Oslo, Norway
aiko.fallas@gmail.com

Sherlock Licorish
Department of Information Science
University of Otago
Dunedin, New Zealand
sherlock.licorish@otago.ac.nz

Jens Dietrich
School of Eng. and Advanced Tech.
Massey University
Palmerston North, New Zealand
j.b.dietrich@massey.ac.nz

Steve Counsell
Department of Computer Science
Brunel University
London, UK
steve.counsell@brunel.ac.uk

## ABSTRACT

This paper investigates how developers discuss *code smells* and *anti-patterns* over Stack Overflow to understand better their perceptions and understanding of these two concepts. Understanding developers' perceptions of these issues are important in order to inform and align future research efforts and direct tools vendors in the area of code smells and anti-patterns. In addition, such insights could lead the creation of solutions to code smells and anti-patterns that are better fit to the realities developers face in practice. We applied both quantitative and qualitative techniques to analyse discussions containing terms associated with code smells and anti-patterns. Our findings show that developers widely use Stack Overflow to ask for general assessments of code smells or anti-patterns, instead of asking for particular refactoring solutions. An interesting finding is that developers very often ask their peers 'to smell their code' (i.e., ask whether their own code 'smells' or not), and thus, utilize Stack Overflow as an *informal, crowd-based* code smell/anti-pattern detector. We conjecture that the *crowd-based* detection approach considers contextual factors, and thus, tends to be more trusted by developers over automated detection tools. We also found that developers often discuss the downsides of implementing specific design patterns, and 'flag' them as potential anti-patterns to be avoided. Conversely, we found discussions on why some anti-patterns previously considered harmful should not be flagged as *anti-patterns*. Our results suggest that there is a need for: 1) more *context-based* evaluations of code smells and anti-patterns, and 2) better guidelines for making *trade-offs* when applying design patterns or eliminating smells/anti-patterns in industry.

## CCS CONCEPTS

• **Software and its engineering → Software evolution**; *Design patterns*; *Maintaining software*;

## KEYWORDS

Code smells, anti-patterns, mining software repositories, Stack Overflow, empirical study

**ACM Reference Format:**
Amjed Tahir, Aiko Yamashita, Sherlock Licorish, Jens Dietrich, and Steve Counsell. 2018. Can you tell me if it smells? A study on how developers discuss code smells and anti-patterns in Stack Overflow. In *EASE'18: 22nd International Conference on Evaluation and Assessment in Software Engineering 2018, June 2018, Christchurch, New Zealand.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3210459.3210466

## 1 INTRODUCTION

Code smells [9] and anti-patterns [6] reflect design and/or implementation issues in software source code that may potentially have a negative impact on software quality attributes such as maintainability. A large body of research in the area of code smells have therefore focused on smell detection and removal techniques, including the work of Marinescu & Lanza [14], and studies by Tsantalis & Chatzigeorgiou [33] and Moha et al. [17]. However, recent work [21, 38] hints that code smells are not always very useful for identifying problematic code. Moreover, other studies have reported that a major challenge for those using smell detection tools is the problem of too many false positives [8]. We postulate that currently the software engineering community faces a gap in their understanding around the level of criticality behind the concepts of code smells and anti-patterns, and hence, software developers are not well supported.

Developers thus congregate on blogs and similar portals in search of critical discourses around this issue. Stack Overflow, in particular, offers utility for teasing out such issues and promoting awareness among developers. Since its launch in 2008, Stack Overflow has moved from being a simple Q&A platform to a comprehensive repository of developer knowledge. Stack Overflow covers both technical and general discussion on various software development topics. It is also acknowledged as a good source of high-quality code snippets that developers can reuse [30]. As of 18 January 2018, Stack Overflow comprised around 15 million questions asked and discussed by over 8.3 million users[1]. Given its emergence as a popular and comprehensive (developer-based) knowledge repository in software engineering, we assert that Stack Overflow is a relevant source of information for investigating the state of developers' *understanding* and *praxis* regarding code smells and anti-patterns. In fact, there are other similar popular forums that developers use to discuss code smells and anti-patterns such as Stack Exchange Software Engineering[2] and Code Project[3]. However, Stack Overflow is seen to be much larger in size in terms of the number of users that contribute to this platform compared to the other available forums. In this study we use Stack Overflow as our "first-step" towards studying code smells and anti-patterns based on discussions established between developers on-line. We plan to include other sites in the near future such as Stack Exchange Software Engineering and other similar sites and blogs to provide more comprehensive assessments of developers' views on smells and anti-patterns. Combining results from multiple sources is strongly required to provide a better and more comprehensive understanding of developers perception of code smells and anti-patterns.

The goal of this study is to examine how the topics of *code smells* and *anti-patterns* are discussed among developers in Stack Overflow. In particular, to obtain a better understanding of *how smells and anti-patterns are understood, perceived by developers* and *what corrective actions* (if any) *developers take to deal with them.* To the best of our knowledge there are only two prior studies that have sought to understand the perceptions of developers regarding code smells. Yamashita & Moonen [38] conducted a survey with 85 software developers to understand what developers think of code smells. Palomba et al. [21] studied perceptions of 12 code smells in Java with 19 developers and 15 postgraduate students. This latter study did not consider smells other than the 12 selected ones, or languages other than Java. Also, the study was done in a controlled environment (developers were asked to identify smells in given code). Our study complements these works by: 1) addressing a larger set of developers (in that Stack Overflow offers an opportunity to study a large number of posts by many developers), and 2) studying the notions of code smells and anti-patterns across different programming paradigms and languages. More interestingly, we study practitioners' logs where socially desirable responding is reduced, thereby providing triangulation for earlier works.

The remainder of this paper is structured as follows: Section 2 presents related work. The study design is presented in Section 3. Section 4 presents and discusses the results. We present some key observation in Section 5, alongside concrete examples of the Q&A postings. The implications of this study are then presented in Section 6, followed by the threats to validity in Section 7. Finally, Section 8 presents our conclusions and future work.

[1]http://stackexchange.com/sites
[2]https://softwareengineering.stackexchange.com
[3]https://www.codeproject.com

## 2 RELATED WORK

### 2.1 Studies on Stack Overflow

In recent years studies have considered a range of issues faced by developers by mining Stack Overflow. Previous studies have used Stack Overflow data to study users' behavior and topic trends [19, 20, 26, 35]. Others have applied Topic Modelling techniques to analyse Stack Overflow questions to categorise topics and discussions [2, 4]. Several other works have analysed specific posts related to particular topics, such as web development [3], mobile development [15, 25], cryptography APIs [18] and mobile app energy consumption [23].

Rosen and Shihab [25] conducted a study to investigate which questions and topics were related to mobile development on Stack Overflow. Their study found the most popular topics of discussion in relation to mobile development to be app distribution, APIs and data management. Reboucas et al. [24] mined Stack Overflow to investigate how developers use the Swift programming language. In particular, the study focused on the problems faced by developers when using Swift.

In work more closely related to our own study, Choi et al. [7] studied how the topic of code clones (also considered to be a type of code smell) is discussed on Stack Overflow, with an emphasis on one particular question: which languages were the most widely discussed in relation to code clones. The study checked all *tags* associated with "clones" in Stack Overflow and found that C#, Java and C++ were the languages associated with the most questions on code clones.

### 2.2 Studies on Code Smells and Anti-patterns

The impact of code smells on developers' daily tasks has been discussed (mostly) in controlled settings. For example, several works have studied the impact of code smells on software quality using a group of developers working on a specific project [22, 27, 28, 36]. Most reported studies have focused on investigating the impact of smells on software quality attributes (such as defect-density [11, 13] and code readability and understandability [1]); when such an impact exists, a set of refactorings is provided.
Other studies have also investigated the impact of different forms of code smells and anti-patterns on software quality, such as architectural smells [8, 10], test smells [5, 31] and spreadsheet smells[12].

## 3 STUDY DESIGN

### 3.1 Research Questions

This aim of this work is to obtain a fuller understanding of developer perception of code smells and anti-patterns and what actions they undertake to deal with them. By determining what developers think about the impact of smells and anti-patterns, we can calibrate our research towards those that are of most concern to developers. To achieve the goals of this study we address three research questions, as follows.

**RQ1. Which code smells and anti-patterns are the *most actively discussed* in Stack Overflow?**

To answer this question we examine the numbers of posts that discuss specific code smells or anti-patterns (a list of all identified code smells and anti-patterns is presented in Table 1).

**RQ2. What are the concepts/definitions involved in the questions that developers tag with "*code smell*" or "*anti-pattern*"?**
To answer this research question we traverse Stack Overflow questions with the following *Tags*: "code smell" or "anti-pattern". We read through all such questions and their associated answers (and comments) to *manually* capture the concepts and definitions involved in the discussions.

**RQ3. What are the *corrective actions* suggested to deal with or resolve a given code smell or anti-pattern?**
In addressing this question we manually analyse 100 most popular questions in our dataset (and their associated answers and comments) to identify the names and types of smells and anti-patterns discussed and also the actions taken to deal with the smells/anti-patterns. We also investigate whether refactoring recommendations are suggested by developers to deal with a given code smell or anti-pattern.

## 3.2 Data Extraction

We mined Stack Overflow data through the Stack Exchange Data Explorer web interface[4]. Stack Exchange releases "data dumps" of its publicly available content (including Stack Overflow) and make them accessible through Data Explore. Using the on-line data explorer provides up-to-date access to the latest data dump by Stack Overflow. We ran an SQL query to extract the data relevant to our needs from this data dump. Our data was obtained on 21-10-2016. Our SQL query is available online for replication and validation purposes[32].

In Stack Exchange data sets, both *questions* and *answers* are considered as *Posts*, and are all added to the *Posts* table. Given that we are interested primarily in the questions asked by developers, we run our query to search for questions only. This can be identified through the *PostTypeId* field in the *Posts* table[5]. Questions are given a *PostTypeId* value of 1, where answers are given a value of 2. Therefore, our designed SQL query searches through all posts, but returns only questions. Note that, for the purpose of this analysis, we mined for code smells or anti-patterns related to source code and software design - other forms of smells and anti-patterns were excluded from this analysis, such as DB or requirement smells.

Searching exclusively via *Tags* can be ineffective [4]. There are several disadvantages of using *Tags* as the only method to determine whether a post is related to a topic. A user who creates a question could be unsure about the title of the most appropriate tag for their discussion, which can lead to the use of *incorrect* or *irrelevant* tags. For example, we observed that some developers used the tag 'Clone' for questions related to three unrelated topics: Code Cloning (i.e., code duplication), Git cloning feature and 'object cloning' in Object Oriented languages (such as the `clone()` method that is used to duplicate objects). Another issue with user-defined tags is that many users tend to add as many tags as possible (Stack Overflow allows up to 5 tags) to increase the number of views and potentially increase the likelihood of receiving answers quickly [3]. Thus, while tags can be helpful to capture questions related to code smells and anti-patterns, using tags *exclusively* may miss important questions on this topic. Therefore, we decided to mine the main *Posts* (the body of each question) to extract our data. However, we still used tags as an initial indicators of the extent of discussions (RQ1) and also to check for what developers declared as a code smell or an anti-pattern question (RQ2). In our search, we used several variations of terms that refer to code smells or anti-patterns, including: "code smell", "bad smell", "anti pattern", "anti-pattern" and "technical debt". Given there is a chance that a user might ask a question about a specific code smell (e.g., Blob) without including in the question terms such as "code smell" or "anti-pattern", we also included a list of code smells and anti-pattern names in our search. Those we included were extracted from nine relevant studies on this topic [9, 11, 13, 21, 22, 34, 36, 37, 39], including the first work on code smells by Fowler [9] and the systematic review by Zhang et al. [39].

We first combined the list of smells and anti-patterns from all nine studies. We then checked how frequently each was used across the above mentioned studies. We decided to include in our search query all smells that appeared in at least two studies. The resulting list thus included 34 unique code smell and anti-pattern terms. We believe this would enable us to be inclusive and reduces the chance of missing any important relevant questions on the topics of interest. When running the SQL query, however, we encountered difficulties in retrieving results when we included all of the smell and anti-pattern terms and names. After a number of pilot runs we noticed that problems arose when long strings were included in the query (and in particular, those containing more than two words). In contrast, the query would work well after the removal of any terms comprising more than two keywords. Therefore, we chose to remove the following keywords from the query: "Long Parameter List", "Class Data Should be Private" and "Alternative Classes with Different Interfaces". The final list of terms, smells and anti-pattern names included in our search is shown in Table 1. A description of these smells and anti-patterns is provided externally. Our full dataset is available for validation and replication studies [32]. In total, our search retrieved 17,126 questions posted between August 2008 and October 2016. We then ran several filtering steps to clean the data and eliminate false-positive results. Our filtering approach is explained in the following section.

## 3.3 Data Filtering

*Inclusion/Exclusion criteria.* We included any questions that referred to source code smells or design anti-patterns. We excluded questions related to other forms of smells and anti-patterns, such as those associated with databases and spreadsheets.

*Removal of duplicates.* After consolidating the data, we found seven duplicate results, which were then removed.

*Automated and manual filtering.* We ran a manual check on the first 50 questions (sorted by Creation Date) to assess the form of the results and their suitability for our intended analyses. For the selected questions, we checked the *Tags*, *Title* and *Body* of each. We identified 28 questions as false-positive results (i.e., questions that were not directly related to code smells or anti-patterns). These questions were all related to the term *Blob*, which could refer to the relevant term "Blob Class", but which is also used as a type in relational databases. The 28 questions identified as false positives

[4]http://data.stackexchange.com/stackoverflow
[5]https://goo.gl/8Mz6da

**Table 1: Terms that we included in our mining**

| General terms | Specific smells terms (cont'd) |
|---|---|
| Code Smell | Code Duplication |
| Bad Smell | Code Clone |
| Anti-Pattern | Refused Bequest |
| Anti Pattern | Data Class |
| Technical Debt | Switch Statements |
| **Specific smells terms** | Case Statements |
| Feature Envy | Message Chain |
| Lazy Class | Middle Man |
| Speculative Generality | Parallel Inheritance |
| Data Clumps | Blob |
| Shotgun Surgery | God Class |
| Divergent Change | Brain Class |
| Large Class | Complex Class |
| Long Method | Spaghetti Code |
| Duplicated Code | Temporary Field |

belonged to three categories: *database* questions (SQL questions that considered the use of blob as a datatype that stores a Binary Large Object as a column in a row), *flash* related posts, and questions related to the Google app engine. We therefore ran another SQL query to filter out these questions through the Body and Tags fields. For the body of the questions, we added the term "*class*" after "*blob"* in order to make the query more precise. We also filtered out *Tags* related to the three categories described above. After manual verification, we identified five *Tags* as irrelevant: *blob*, *SQL*, *database*, *flash* and *Google app engine*. After applying our automatic filtering approach, the total number of questions fell to 3,109 questions.

*Manual validation.* We followed up the automatic filtering with a careful manual validation, checking the *Tags*, *Title* and *Body* of each individual question to exclude all irrelevant questions.

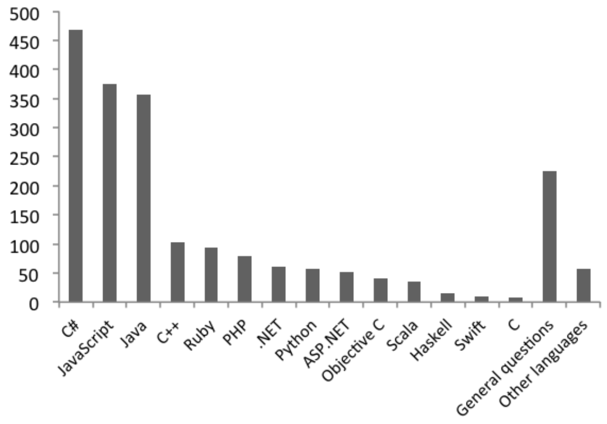
Manual validation was used to complement the primary automatic search. Automatic search can produce large numbers of False Positives and given the size of the dataset we could not be 100% sure if we included only the correct posts. Manual verification of the data helped to reduce the number of false positives and ensured that we had the right set of posts for the analysis.

This process was performed by two researchers, the first author of this paper and a PhD student from the same institution. The second coder was introduced to the work and then given examples of how questions should be categorised. Before doing the actual coding, the two coders categorised 50 posts together. Once they had finalised the coding scheme, the first coder categorised 1,200 questions while the second coder categorised another 1,909 posts. We followed this step with a careful cross-validation process where the coders validated 100 of each other's coded posts. The selection of the 100 posts from each coder was random. The two coders discussed each classification they performed, and the process resulted in over 90% agreement between the two coders. We also conducted another manual check for RQ2 and RQ3 (explained in Section 4). For RQ2, we manually analysed a set of 206 questions that were tagged as either code smell or anti-pattern. The classification for this question was done by the first author and a selected number of posts were cross-validated by one other author for each list (i.e., we cross-validated 24 posts from the code smells list and 38 questions from the anti-pattern list). For RQ3, we also manually analysed 100 posts (explained in detail in Section 4), mainly by the first author, but when the first author was not able to classify the post it was then referred to one of the other co-authors for verification. In total, we cross validated 15 posts and reached 100% agreement on the classification of these posts between both coders.

*Classification.* As part of the filtering process, we manually classified questions based on the targeted programming language. For this analysis, we did not depend solely on the tags used as we noticed that some questions were not tagged based on the programming language used, but rather for a specific feature or framework. For example, some JavaScript questions were tagged with AngularJS, a popular JavaScript framework. We found that there were large numbers of posts that ask programming-independent questions or general questions about code smells or anti-patterns. We classified such questions as "general"- that is, a question that is not associated with a specific programming language. Note that while performing the classification of posts and reading through the posts and discussions, we noted down and then discussed different phenomena that we observed. This has led to several interesting findings, which will be presented in Section 5.

Given the nature of Stack Overflow, there is a possibility that some of the questions asked on the forum might be closed by the moderators for one reason or another. The nature of questions about code smells and anti-pattern can be very broad and opinion-based, and those sort of questions are not favored by Stack Overflow moderators. Therefore, some interesting questions might be the subject of closure by the moderators. However, we argue that some interesting programming related discussions can be established because they are directly or indirectly related to code smells or anti-patterns, and such questions and discussions should be investigated to see if users generally share the same view about the impact of code smells or anti-patterns. In general, code smells and anti-patterns should not be seen as a separate topic from other common programming topics. Many concrete programming problems arise because of specific code smells or anti-patterns, and those discussions are widely welcome on Stack Overflow. Consequently, we checked how many closed questions were contained in our dataset to ensure that we did not deal with large set of irrelevant, or largely outdated, posts. Of the final list of questions included in the analysis, only 115 questions were marked as closed for various reasons, representing less than 4% of the total of number of questions in our dataset. For those closed questions, the average time between the time the question was posted and the time the question was closed was 11 months. Although those questions were closed, they still had a relatively high Score (as high as 156 in one case) and view count (as high as 90304). We thus acknowledge that there is only a small set of questions in our dataset that are marked as closed, and even though those questions are closed, they have stayed active for a while and are actively viewed by users, thus, indicating their relevance to the issues under consideration.

**Figure 1: Number of questions on code smells and anti-patterns asked on Stack Overflow considered by programming language**



## 4 RESULTS AND DISCUSSION

We first checked which programming languages were commonly referred to in questions about code smells and anti-patterns. Figure 1 shows that C#, JavaScript and Java were the languages with most questions on code smells and anti-patterns, constituting 59% of the total number of questions on these topics. The data also shows that there are relatively large numbers of code smell and anti-pattern questions on .NET technologies (not only C#). However, there is, equally, a substantial proportion of questions (around 11%) which are language-independent (i.e., do not relate to a specific programming language). We also noticed that there was a relatively low incidence of questions involving functional programming languages such as Haskell compared to mainstream OO languages. This is mainly because of the fact that the majority of smells that are studied in the literature (and also included) are related to OO languages. However, some of these smells are somehow general and can be relevant to any programming paradigm (i.e., *code clone* smell or *service locator* pattern).

These results are in line with the language Popularity of Programming Language Index (PYPL)[6], in that 8 of the programming languages that we found in our analysis are represented in the top 10 languages in the PYPL list, although in a slightly different order (i.e., top 5 languages in the PYPL list are Java, Python, PHP, C# and JavaScript). This is also in line with the language popularity statistics provided by Stack Overflow[7], which postulates that JavaScript, C#, Java, C++, PHP and Python are among the top 10 languages most commonly used by developers.

To assess the level of attention paid by developers to these topics over time, we looked at temporal trends in posts on code smells and anti-patterns in Stack Overflow. We divided the data into bi-yearly periods (1st half: Jan-June, 2nd half: Jul-Dec). The results (presented in Figure 2) show that there has been a steady increase in the numbers of questions asked by developers over time, growing from 33 in the 2nd half of 2008 to over 410 in 2015. More than 360 questions have been asked in 2016, though the figure is not complete as it does not cover the whole year (only until October, when the data was retrieved). We expect that the number of questions in 2016 will surpass the 2015 figures. While the trend is clearly evident, such an increase could be a result of the growth in the number of Stack Overflow users over the period in question.

[6] http://pypl.github.io/PYPL.html
[7] https://goo.gl/wktwTc

**RQ1. Which code smells and anti-patterns are the *most actively discussed* in Stack Overflow?**

To answer this question, we mined all questions in our dataset to check if a particular code smell or anti-pattern (from the list of smells and anti-patterns in Table 1) had been discussed in a post. For the purposes of analysis, we grouped smells of a similar nature, i.e., smells and anti-patterns that had the same meaning and interpretation, as follows: 1) *Blob*, *God Class* and *Brain class*, and 2) *Duplicated Code* and *Code Clone*. Due to space constraints, we provide detailed results for this question externally, which also include results of how frequent each smells/anti-patterns were discussed in popular programming language [32]. The key results for this question are shown in Figure 3. We found relatively few questions that used the same names of smells/anti-patterns that we included in our search (representing approximately 10% of the total number of posts we retrieved). Of this list, we found that the smells and anti-patterns *Blob*, *Duplicated Code* and *Data Class* are by far the most frequently discussed on Stack Overflow. Most of the other smells that we included in our search were rarely discussed, e.g., *Refused Bequest* and *Message Chain*. We found that no questions addressed the following smells or anti-patterns by name: *Speculative Generality*, *Data Clumps* and *Divergent Change*. While the number of posts that used the same terms as in our search is relatively low, we believe this is an important finding in itself. It could be that either developers do not discuss these smells and anti-patterns as widely as we expected, or that they use other terms to refer to the same smells/anti-patterns. We discuss this issue further in Section 7.

We also investigated the popularity of questions asked about code smells and anti-patterns by considering the *View Count* and *Score* statistics obtained from Stack Overflow. We first checked the rank correlation between *View Count* and *Score* values using

**Figure 2: Temporal trends in code smell and anti-pattern questions**

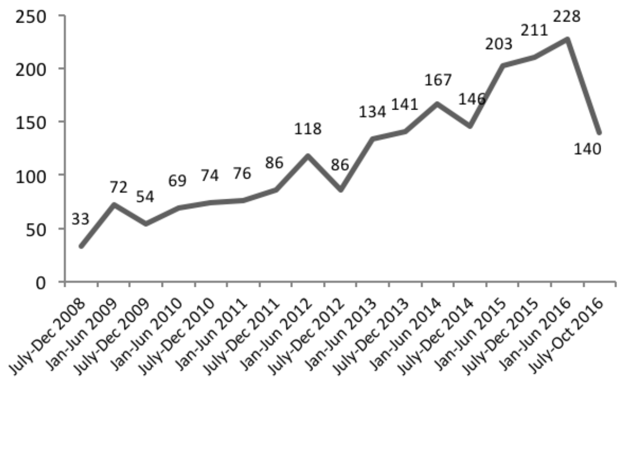
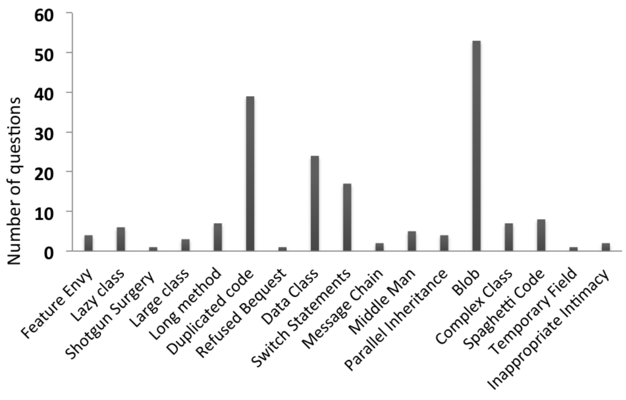


**Figure 3: Number of questions for each individual code smell**

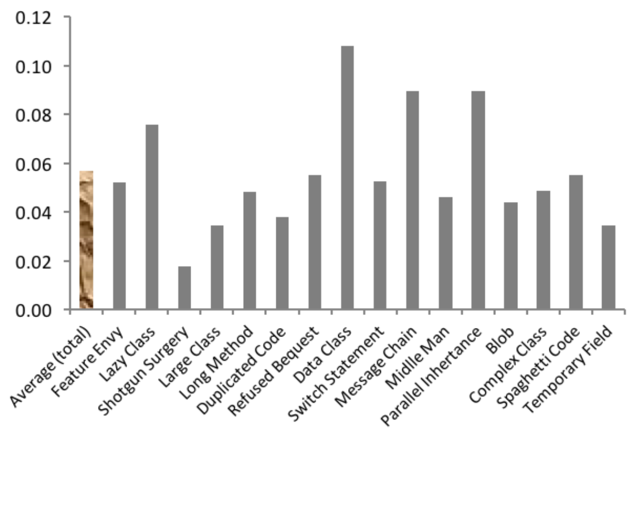


Spearman's rho $\rho$. We found that *View Count* to be significantly correlated with *Score* ($\rho = 0.61$, $\alpha < 2.2\text{e-}16$). We normalized all *View Count* and *Score* values using Feature (Min-Max) Scaling, according to the following formula:

$$X_{nor} = \frac{X - X_{\min}}{X_{\max} - X_{\min}} \tag{1}$$

where $X_{nor}$ is the normalized value, X is the original value, $X_{\min}$ is the minimum value in the range and $X_{\max}$ is the maximum value in the range. We then created *a new question popularity metric* by taking the average of the two normalized values of *View Count* and *Score*. We show the results of this analysis in Figure 4. In terms of popularity, we found that questions on *Data Class*, *Message Chain* and *Parallel Inheritance*, followed by *Lazy Class* and *Feature Envy*, were the most popular. Questions regarding other smells/anti-patterns, such as *Shotgun Surgery* and *Large Class*, had low popularity in comparison.

**Figure 4: Popularity for each individual smell and anti-pattern**



*Taken overall, our results show that* ***Blob****,* ***Duplicated Code*** *and* ***Data Class*** *are the most frequently discussed smells in Stack Overflow. For certain smells and anti-patterns in our list (Table 1), we did not find any questions being asked about them. These smells/anti-patterns are:* ***Speculative Generality****,* ***Data Clumps*** *and* ***Divergent Change****.*

**RQ2. What are the concepts/definitions involved in the questions that developers tag with "*code smell*" or "*anti-pattern*"?**

To answer this research question we examined Stack Overflow questions that were tagged with either *code smell* or *anti-pattern.* In total, we found 206 questions in this category (i.e., 80 questions with the tag *code smell* and 126 questions tagged with *anti-pattern*). We then removed one instance of eight duplicate questions (that is, questions that were tagged with both *code smell* and *anti-pattern*). The results for each of the two tags were then analysed separately. We manually reviewed the body of these questions and their answers/comments and then categorised each question based on the name of code smell or anti-pattern that the question was discussing. All general questions that were asked (i.e., where no specific smell or anti-pattern was mentioned) were classified as "General" questions. We excluded questions that we considered to be irrelevant to code smells or anti-patterns (i.e., where a question is tagged as a *code smell* or *anti-pattern* question but did not specifically discuss code smells or anti-patterns). This process was done by the first author of the paper and a selected number of posts (from each list) were cross-validated by one other author for each list (i.e., we cross-validated 24 posts from the *code smell* list and 38 questions from the *anti-pattern* list). In total, we marked 6 questions from the *code smells* list and 20 questions from the *anti-pattern* list as irrelevant or not related. We found that some questions discussed more than one code smell or anti-pattern. In such cases, we include all the names of these smells or anti-patterns. For the questions assigned the *Tag* "Code Smell", we found that 11 of these were general - they were not specific to a particular code smell or anti-pattern. The nature of these questions ranged from general knowledge to tool-specific questions. The majority of the other questions revolved around various forms of code smells, including production-code and test smells. While the majority of smell types we found were known (such as *God Class/object*, *Type Checking*, *Spaghetti Code* etc.), we found new so-called 'code smells' being discussed between developers but which might not be covered by current definitions of code smells. Such smells are also not widely discussed or studied by researchers. This includes the terms: *Useless Override*, *Nested Try-Catch block* and *Exploit Polymorphism*. We also found that there were several 'smells' that represented violations of design patterns or coding "best practice". For example, several smells suggested by developers represent violations of design principles such as *Don't Repeat Yourself (DRY)*, *Liskov Substitution Principle (LSP)*, *Single Responsibility* and other *SOLID* principles [16]. For the questions with the "anti-pattern" *Tag*, the majority of questions involved a wide range of anti-patterns, with 15 questions discussing their general aspects. A number of these posts discussed the implications of violating basic design principles (such as DRY and Open-Closed Principles) and suggested these violations as anti-patterns. Other questions discussed the downfalls of certain design patterns (i.e., how some design patterns could potentially appear as anti-patterns). The *Service Locator* and *Singleton* design patterns are particularly widely discussed in the context of anti-patterns. Such questions discussed the downsides of these patterns and suggested that certain implementations of these patterns could result in anti-patterns.

One additional finding was that developers do not always distinguish between the terms *code smell* and *anti-pattern* - rather, they use the two terms interchangeably to refer to the same or similar design or implementation issues. Several questions that were tagged with *code smell* in fact discussed *anti-patterns* and vice versa. For example, developers tagged questions about *LSP* and *Service Locator* as code smells and questions about *Duplicated Code* and *Ravioli code* as anti-patterns.

*There is a lack of consistency in the definitions used in posts tagged as **Code Smell** and **Anti-Pattern**. With regard to their underlying meanings, both terms are used interchangeably (i.e., in the same text to refer to code smells as anti-patterns, or vice versa). The violations of certain design principles such as **LSP** and **DRY** are widely described as anti-patterns. Many questions discussed the downsides of some well-known design patterns (such as **Service Locator** and **Singleton**) and suggested them as potential causes of anti-patterns.*

**RQ3. What are the corrective actions suggested to resolve a given code smell or anti-pattern?**

For this question, we manually analysed the top 100 questions (based on the popularity metric described in formula (1)) to check the actions taken (or recommended) by developers to deal with smells and anti-patterns. We chose only the top 100 posts so that we could carefully analyse these questions and all their answers with the available resources. Our analysis comprised three main aspects: 1) the name or type of the smell or anti-pattern, 2) the action recommended in the accepted answer, and 3), if there were some suggested refactorings, the name of the refactoring operation suggested in the answers. For the actions recommended, we focused mainly on accepted answers (i.e., an answer that was accepted by the person who asked the original question). For the purpose of the analysis of this question, we categorise *Actions* into one of the following four categories: (1) **Fix**: recommendations made to fix the code, e.g., remove the smell or anti-pattern by refactoring), (2) **Capture**: detect the smell/anti-pattern but no direct refactoring recommendations are given, (3) **Ignore**: recommend to ignore taking any action, e.g., assumes that the smell or anti-pattern has no bad side effect, and (4) **Explain**: if the question asks for information and the accepted answer provides only an explanation of the smell/anti-pattern, e.g., if the answer provides only an example of why something is considered a smell or anti-pattern. Of the 100 questions we found five that were not relevant to the topics of smells or anti-patterns (i.e., it uses terms like "smells" or "anti-patterns" but does not necessarily discuss anything related to these topics), and therefore we decided to exclude these. In total, we include 95 posts that we analysed manually. In general, our results show that most questions asked for general opinions on smells or anti-patterns, and sought only an explanation rather than a solution.We found that 59 of these questions fell into the *Explain* category. Only around 25% of the posts provided some fixing strategies to deal with smells/anti-patterns. The majority of questions (56%) did not provide any refactoring recommendations. For those questions that were categorised as *Fix* or *Capture*, a number of refactoring operations were typically suggested in the answers. However, we observed that most of the refactoring recommendations (17%) were code-based (i.e., provide a coding example via a code snippet). In most cases, no specific refactorings were identified by name. Some answers (in 7 different questions) recommended the implementation of certain design patterns, such as *Dependency Injection* as a potential refactoring solution for certain smells/anti-patterns questions.

In comparison, we found only a few posts that fell into the *Capture* (5 questions) and *Ignore* (7 questions) categories. Due to space limitations, we provide the full results to this question externally [32].

*For the top 100 posts, most answers provide explanations of a particular smell/anti-pattern, without providing specific refactoring recommendations. Only 25% of these posts have some fixing recommendations (i.e., refactoring) in their accepted answers. Even when such recommendations are provided, no specific names are given to such operations, as most of these refactorings are shown as code snippets.*

## 5 OBSERVATIONS FROM MANUAL ANALYSIS

We made a number of observations based on our analyses of the questions examined in this study, which we present alongside some sample illustrative quotes (for reference we also include Post IDs)[8].

**Observation 1**: *Developers use the terms "code smell" or "anti-pattern" rather liberally (i.e., everything that is not to their liking "stinks").* There are questions that use the term "code smell", but do not specifically refer to an actual code smell. They rather use the term to express their disgust or repugnance about their code. Here is an example of such usage (#8921639):

> "*... **my code smell is that they are not good enough.** Is there any better and faster way?"*

Another example is shown here (#29841845):

> "*... I know this **code smells badly**, but my question is only about safeness of this code.*"

Similarly, the term (and also the tag) anti-pattern is used in many ways. It does not always refer to design issues, but rather to any 'pattern' issues in the source code, in the configuration files or even in the tools. Some developers discuss the usefulness of some of the common design patterns. In contrast, some of the design patterns (or their implementations) have been suggested as possible cause of anti-patterns. For example (#31516310):

> "*... before i implemented this, i went out (sic) a did a bit of research about whether it was a good idea or not, and all the information i could find (sic) we statements calling it an anti-pattern*

[8]To access the question on the web, add the given PostId number to the question URL as follows: stackoverflow.com/questions/"PostId number"

> *but without explaining why. Why is a generic repository considered an anti-pattern?*"

**Observation 2**: *Developers more often ask whether something is a smell or an anti-pattern or not, rather than referring to a particular smell or anti-pattern.* Much of the code designated as "smelly" or a design that is called an "anti-pattern" are presented generally, without mentioning any specific names for the type of code smell or anti-pattern. For example, in one of the posts (#7190450) a user wrote the following:

> "*. . . I spent already too much time on this problem and my current approach uses a convenience method to add the: username parameter. Nevertheless, I think using this method all over the place just to add an entry stinks (bad code smell). . .* ".

**Observation 3**: *Developers ask about the trade-offs of the usage of known design patterns.* There are questions asked about the *overuse* of certain design patterns and the implications associated with that. We noticed that such questions explain how the 'overuse' of certain design patterns can potentially turn them into anti-patterns. For example, in post (#20663466) a user asked the following question about the *Promise* design pattern:

> "*. . .* ***Is it an anti-pattern to extend promises with extra functions?*** *I have this example of a service that does API calls to Facebook. . .* ***What are the cons of such an implementation?***"

Here is also another example of how the implementation of one design principle, i.e., Single Responsibility Principle (SRP), can lead to an anti-pattern, i.e., Anemic Domain Model (#1399027):

> "*I'm trying to improve my understanding of how to apply SRP properly. It seems to me that SRP is in opposition to adding business modelling behaviour that shares the same context to one object, because the object inevitably ends up either doing more than one related thing, or doing one thing but knowing multiple business rules that change the shape of its outputs. If that is so, then* ***it feels like the end result is an Anemic Domain Model . . . Yet the Anemic Domain Model is an anti-pattern.*** *Can these two ideas coexist?*"

*Dependency Injection* and *Service Locator* design patterns appear as the most discussed anti-pattern topics, including their use in different platforms and scenarios. Most of the discussions are centered on whether they are design patterns or anti-patterns. In many cases, developers provide some scenarios (mostly via code snippets) where they discuss the negative impact of using and implementing these patterns in their application. Here is an example of a *Service Locator* question (#22795459):

> "*Recently I've read Mark Seemann's article about Service Locator anti-pattern. . . two main reasons why ServiceLocator is anti-pattern: API usage issue . . . (and) Maintenance issue. . . I'm not trying to defend Service Locator approach. But this misunderstanding make me think that I'm losing something very important. Could somebody dispel my doubts?*"

In addition, the *Singleton* pattern was widely discussed with many developers considering certain implementations of this pattern as a potential anti-pattern. In other questions, some developers wonder whether anti-patterns or code smells truly have an impact on their code. In particular, we found questions on why developers should not concern themselves with the presence of code smells or anti-patterns at all, in certain scenarios. We also noticed questions which *doubt* the practicality of certain known design patterns such as *Singleton* and *Service Locator*. An example of such a question is shown here (#568365):

> "*Am I the only one that sometimes take the seemingly easy, but wrong, way out of certain design situations? I'll admit I've made my share of questionable Singleton objects. Besides that, I've been known to make a God object or two to make things seem easier.* ***Do you ever use an anti-pattern even though you know you shouldn't?***"

Here is another example, from post (#16944712):

> "*. . . You could argue that this design ambiguity is a code smell but sometimes it is unavoidable at least as far as pragmatism allows*".

Many questions on the forum discuss smells or anti-patterns that are language- (or even library-)specific. Usually users provide a scenario or an example from a specific programming language about a smell or an anti-pattern, and then ask for confirmation of their initial assessment. An example from this category of questions is shown in the following text, which is specific to Java (#7813662):

> "*. . . the point came up that use of the Thread.Sleep() method is a code smell. I have a hard time believing there is no use of this method that doesn't indicate that you're doing something wrong. . . . why would Thread.Sleep be such a terrible line of code to use in any circumstance that it would smell so?*".

The following is another example of an anti-pattern specific to React.js (a JavaScript library) (#38421693):

> "*. . . I was searching for an answer to another problem and I found this answer (link) I'm just curious why is it that modifying its own props is an anti-pattern, and why isn't it that modifying its own state is not and (sic) anti-pattern?*".

**Observation 4**: *Developers ask general questions about code smells and anti-patterns.* Some developers ask general or open questions about the *notion* of smells and anti-patterns. These questions could be classified as *knowledge inquiry* as they are not discussing a specific scenario or programming language. An example is evident in the following two posts (#851412), (#980601):

> "*I'm looking to see how other programmers find anti-patterns, "code-smells". . . what things start*

> *setting you off when you're looking at code that tells you, something has gone wrong here?"*
>
> "*. . . Can anybody explain to me in simple words that what an anti-pattern is? What is the purpose? What do they do? Is it a bad thing or good thing?".*

**Observation 5**: *Developers ask about tooling and support for code smells and anti-patterns.* Several questions discuss code smells or anti-patterns in the context of tools or IDEs. Some developers ask questions to look for help with tools to detect code smells or anti-patterns, or look for specific configurations for tools or IDEs. For example, a user asked the following question (#7036634):

> "*I am using NetBeans for development. . . what I am looking for is something: - free (also for commercial use) - for Windows (or cross-platform) - must be able to scan a project by starting from a given "main" file (usually index.php) and following include/require. . . Basically, I am just looking to make "code smells" noticeably & easy to examine/fix.".*

We also observed that developers frequently provided answers that contained refactoring solutions to the code smells and anti-patterns that were submitted in the questions. However, a large proportion of such refactoring solutions (around 45%) were suggested through code examples (i.e., provided as code snippets). When available, developers *mostly* did not provide *specific* names to the refactoring implementations that they provided as solutions.

In the following section we discuss some of the implications of these findings for researchers and practitioners.

## 6 IMPLICATIONS FOR RESEARCH AND PRACTICE

The research in this paper has a number of implications for both research and practice. First, there is unclear evidence on how harmful developers think smells and anti-patterns are; the standard texts on both make no judgment on which are more likely to cause problems in a system and, as far as we know, there is no other evidence to support actual developers' opinion on smells and anti-patterns. Perhaps the academic community perceives a problem caused by smells or anti-patterns that simply does not exist in industry, or at least is not considered a pressing problem. If that is the case, then more research is needed to assess *the true impact* of code smells and anti-patterns, and more collaboration with industry is strongly needed to achieve this purpose. Second, the use of academic tools for detecting code smells in academic empirical studies is mostly based on open-source programs, but perhaps as well as reporting these results, academic studies should focus on the usefulness and viability of these tools in industry. In those respects, our *Observation 5* hints that Stack Overflow can be used as an arena to expose, test and tailor methodologies/tools coming from academia. At the very least, we should be exploring ways in which tools/methods can be tailored to what industry needs specifically, rather than a "one size fits all" approach which is often not an effective use of developer time.

Third, it seems that the "badness" of a smell is very much in the eye of the beholder. A consistent problem with previous studies of code smells and anti-patterns is that the threshold of *what is* and *what is not* a smell or anti-pattern is largely subjective in nature. Only a developer can really judge the extent of "smelliness" in a piece of code and decide to do something about it. This judgment may also be *application domain* specific. To attempt any generalisations about the impact of a code smell is clearly flawed if that is the case.

Finally, there appears to be considerable confusion about what an anti-pattern is, *vis-a-vis* code smells. Our results reveal that the meaning associated with concepts like smell and anti-pattern differs significantly between discussions and users.

This is not surprising if one considers the meaning of those concepts defined by how they are used in context. These contexts (time, experience of participants, programming language, etc.) vary widely between discussions. While this might be a valid observation, it is not satisfying from an engineering point of view. Engineering requires normative definitions – for teams to communicate effectively and vendors to create tools that solve well-understood problems. Such normative definitions exist, in seminal work on the topic by Fowler and others. They seem to fail, however, in the sense that there is little shared understanding (and thus, practicality) of those terms.

There are two possible reasons for this: (1) that these normative definitions are not suitable and (2) that users do not have sufficient knowledge about them. We think that there might be some truth to both views. Many of the original smell and anti-pattern definitions might be *outdated.* For instance, users complain about the low precision of smell detection tools, which might indicate that the original smell definitions are too coarse and need to be refined in order to reduce false positives [8].

The number of such issues is likely to have increased, since many smells were defined before new technologies were adopted (e.g., domain specific languages, code generation). This might have changed the impact those smells have (for instance, by creating false positives, or by considering smells and anti-patterns as acceptable trade-offs).

On the other hand, developers discussing smell and anti-pattern topics have also changed. While smells and (design and anti-) patterns were discussed in eclectic circles in the late 1990s and early 2000s, platforms such as Stack Overflow have commoditised this knowledge and brought people into the discussion who do not have the 'same' experience participants in those discussions would have had 15 years earlier. A deeper investigation into these issues is an interesting topic for future research. A possible solution is to revisit and refine some smell and anti-pattern definitions in the context of current best practice, by creating updates like "Smells 2.0". However, it is unclear what an acceptable process to update those definitions would look like and whether the community would accept this.

Therefore, it is important that both industry and academia establish the challenges of determining a proper definition of each and then tackle those challenges. Finally, our results suggest that developers need more context-sensitive research results that can help them make "trade offs" between introducing/removing anti-patterns and code smells.

## 7 THREATS TO VALIDITY

*Search process:* Determining whether a post is related to code smell or anti-pattern topics was based on selected keywords that are used either in the question title or body. There is a possibility that developers use keywords other than those that we used in our mining query, which would introduce a validity threat. To reduce this threat, we first combined a list of code smell and anti-pattern terms that developers and researchers use, as reported in several previous empirical studies. These studies are widely known and some highly cited and include key works in this area such as Fowler's text book on code smells and refactoring. When possible, we also included alternative terms that are also used to refer to code smells and anti-patterns. For example, for the *Duplicated Code* smell we used other alternative terms such as *Code Clone* and *Code Duplication*. As explained in Section 3, searching exclusively through *Tags* can be ineffective, as Tags can be less informative. Therefore, we decided to mine the body of the questions. In this way, we sought to minimize the risk of missing questions that used incorrect or irrelevant *Tags*.

*Popularity metric:* We designed a new metric for measuring the popularity of a question and that takes into account posts' *Score* and *View Count* values. However, this metric can be biased in the sense that it does not take into account the time-frame of the questions (i.e., when the questions were posted). As our analysis does not distinguish between questions based on time, new questions might not have high *View Count* or *Score* and therefore may have low popularity values; they will automatically fail to qualify for the manual analysis set. We have not yet arrived at a suitable treatment for this threat but some form of time-based normalization might be feasible.

*Bias introduced by manual analysis:* Manually performed analysis could introduce biases due to multiple interpretations and/or oversight. However, for the purposes of answering our research questions on Stack Overflow data, we deem it essential to complement quantitative analysis with interpretive research techniques such as "grounded theory" [29] to understand better the *context* of the quantitative data at hand, to identify and describe different phenomena from a qualitative viewpoint and to explore the underlying mechanisms behind the identified phenomena. We are aware that human interpretation introduces bias, but we attempt to account for it via cross-validation involving multiple evaluators and by cross-checking the results from the classification stage, by involving the three first authors of the paper.

*Generalisation of findings from Stack Overflow:* In this work we sought to understand developer perceptions of code smells and anti-patterns by analysing questions asked on Stack Overflow. However, this introduces another threat to validity since the conclusions of this study are drawn from only one platform. While Stack Overflow is acknowledged as the most widely used Q&A website by developers around the world, there are other programming forums, Q&A sites and blogs where developers discuss similar issues (including code smells and anti-patterns). One such example is the other Stack Exchange site *Software Engineering*[9](previously known as "Programmers"). We believe that additional investigation is required in similar online forums as well as through large-scale developer surveys.

[9]https://softwareengineering.stackexchange.com

## 8 CONCLUSIONS AND FUTURE WORK

This study provides insights into what developers think of (and how they feel about) code smells and anti-patterns. The findings of this study should help to guide future research directions in the area of code smells and anti-patterns. The results of this study show that there is growing interest in code smells and anti-patterns from developers.

The findings also show that most discussions on code smells and anti-patterns are centered around popular programming languages, with C#, JavaScript and Java dominating the landscape. We also found that *Blob Class*, *Duplicated Code* and *Data Class* are discussed more frequently than others smells and anti-patterns. Also, we observed that developers widely discuss the trade-offs of the usage of specific design patterns, and, in some cases, some well-known design patterns have been suggested as potential anti-patterns that should be avoided; the *Service Locater*, *Singleton* and *Dependency Injection* patterns are the most discussed of these topics. We also observed that there are discussions on why well-known anti-patterns are *not* considered very harmful and should not in fact be flagged as *anti-patterns*.

In general, our results show that there is a gap between what researchers and developers discuss in terms of code smells and anti-patterns. Developers generally have negative feelings toward code smells and anti-patterns, but the actual impact of these smells or anti-patterns is open to question. However, there are cases where users explain that some smells and anti-patterns are not always *bad* - some posts suggest that they are not necessarily always worrying.

We see a number of avenues for future work in this area. As suggested by one of the observations from this study, we plan to develop more precise, context-sensitive smells and anti-pattern detection tools (leading to fewer false-positive results). Another avenue of research is to work towards providing a unified catalog of smells or anti-patterns (similar to the IEEE Standard Glossary of Software Engineering), which can help to bridge the gap between tool vendors and developers. While this study provides a view on what developers think of the impact of code smells and anti-patterns in practice, more investigation into *why* some design patterns are seen as potential anti-patterns is strongly needed. We also aim to study wider aspects of developer knowledge and opinion on code smells and anti-pattern. Our next target is to study publicly available projects repositories and issue tracking systems.

## REFERENCES

[1] M Abbes, F Khomh, Y-G Gueheneuc, and G Antoniol. 2011. An Empirical Study of the Impact of Two Antipatterns Blob and Spaghetti Code on Program Comprehension. In *15th European Conference on Software Maintenance and Reengineering*. 181–190.

[2] M Allamanis and C Sutton. 2013. Why, when, and what: Analyzing Stack Overflow questions by topic, type, and code. In *2013 10th Working Conference on Mining Software Repositories*. 53–56. https://doi.org/10.1109/MSR.2013.6624004

[3] K Bajaj, K Pattabiraman, and A Mesbah. 2014. Mining Questions Asked by Web Developers. In *11th Working Conference on Mining Software Repositories*. ACM, New York, NY, USA, 112–121. https://doi.org/10.1145/2597073.2597083

[4] A Barua, S. W Thomas, and A. E Hassan. 2014. What Are Developers Talking About? An Analysis of Topics and Trends in Stack Overflow. *Empirical Software Engineering* 19, 3 (6 2014), 619–654. https://doi.org/10.1007/s10664-012-9231-y

[5] G Bavota, A Qusef, R Oliveto, A De Lucia, and D Binkley. 2014. Are test smells really harmful? An empirical study. 20, 4 (may 2014), 1052–1094. https://doi.org/10.1007/s10664-014-9313-0

[6] W. J Brown, R. C Malveau, H McCormick III, and T. J Mowbray. 1998. *AntiPatterns: Refactoring Software, Architectures, and Projects in Crisis.* John Wiley & Sons, Inc, New York, NY, USA.

[7] E Choi, N Yoshida, R G Kula, and K Inoue. 2015. What do practitioners ask about code clone? a preliminary investigation of stack overflow. In *2015 IEEE 9th International Workshop on Software Clones.* 49–50. https://doi.org/10.1109/IWSC.2015.7069890

[8] F. A. Fontana, J Dietrich, B Walter, A Yamashita, and M Zanoni. 2016. Anti-pattern and code smell false positives: Preliminary conceptualisation and classification. In *23rd International Conference on Software Analysis, Evolution, and Reengineering,* Vol. 2016-Janua. 609–613. https://doi.org/10.1109/SANER.2016.84

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

[10] J Garcia, D Popescu, G Edwards, and N Medvidovic. 2009. Identifying architectural bad smells. In *13th European Conference on Software Maintenance and Reengineering.* IEEE, 255–258.

[11] T Hall, M Zhang, D Bowes, and Y Sun. 2014. Some Code Smells Have a Significant but Small Effect on Faults. *ACM Transactions on Software Engineering and Methodology* 23, 4 (9 2014), 33:1–33:39. https://doi.org/10.1145/2629648

[12] F Hermans, M Pinzger, and A van Deursen. 2014. Detecting and refactoring code smells in spreadsheet formulas. *Empirical Software Engineering* (2014), 1–27. https://doi.org/10.1007/s10664-013-9296-2

[13] F Khomh, M Di. Penta, and Y-G Gueheneuc. 2009. An Exploratory Study of the Impact of Code Smells on Software Change-proneness. In *16th Working Conference on Reverse Engineering.* IEEE.

[14] M Lanza and R Marinescu. 2006. *Object-oriented metrics in practice.* Springer. 1–205 pages. https://doi.org/10.1007/3-540-39538-5

[15] M Linares-Vásquez, G Bavota, M Di Penta, R Oliveto, and D Poshyvanyk. 2014. How Do API Changes Trigger Stack Overflow Discussions? A Study on the Android SDK. In *22nd International Conference on Program Comprehension.* ACM, New York, NY, USA, 83–94. https://doi.org/10.1145/2597008.2597155

[16] Robert C Martin and M Micah. 2006. *Agile Principles, Patterns, and Practices in C#.* Prentice-Hall. 768 pages.

[17] N Moha, Y.-G. Guéhéneuc, L Duchien, and Le Meur. 2010. DECOR: A Method for the Specification and Detection of Code and Design Smells. *IEEE Transactions on Software Engineering* 36, 1 (2010), 20–36.

[18] S Nadi, S Krüger, M Mezini, and E Bodden. 2016. Jumping Through Hoops: Why Do Java Developers Struggle with Cryptography APIs?. In *38th International Conference on Software Engineering.* ACM, New York, NY, USA, 935–946. https://doi.org/10.1145/2884781.2884790

[19] N Novielli, F Calefato, and F Lanubile. 2015. The Challenges of Sentiment Detection in the Social Programmer Ecosystem. In *7th International Workshop on Social Software Engineering (SSE 2015).* ACM, New York, NY, USA, 33–40. https://doi.org/10.1145/2804381.2804387

[20] A Pal, S Chang, and J Konstan. 2012. Evolution of Experts in Question Answering Communities. In *6th International AAAI Conference on Weblogs and Social Media.*

[21] F Palomba, G Bavota, M D Penta, R Oliveto, and A D Lucia. 2014. Do They Really Smell Bad? A Study on Developers' Perception of Bad Code Smells. In *30th International Conference on Software Maintenance and Evolution.* 101–110. https://doi.org/10.1109/ICSME.2014.32

[22] F Palomba, G Bavota, M D Penta, R Oliveto, D Poshyvanyk, and A De Lucia. 2015. Mining Version Histories for Detecting Code Smells. *IEEE Transactions on Software Engineering* 41, 5 (5 2015), 462–489. https://doi.org/10.1109/TSE.2014.2372760

[23] G Pinto, F Castor, and Yu D Liu. 2014. Mining Questions About Software Energy Consumption. In *11th Working Conference on Mining Software Repositories (MSR 2014).* ACM, New York, NY, USA, 22–31. https://doi.org/10.1145/2597073.2597110

[24] M Rebouças, G Pinto, F Ebert, W Torres, A Serebrenik, and F Castor. 2016. An Empirical Study on the Usage of the Swift Programming Language. In *23rd International Conference on Software Analysis, Evolution, and Reengineering,* Vol. 1. 634–638. https://doi.org/10.1109/SANER.2016.66

[25] C Rosen and E Shihab. 2016. What Are Mobile Developers Asking About? A Large Scale Study Using Stack Overflow. *Empirical Software Engineering* 21, 3 (6 2016), 1192–1223. https://doi.org/10.1007/s10664-015-9379-3

[26] V S Sinha, S Mani, and M Gupta. 2013. Exploring activeness of users in QA forums. In *2013 10th Working Conference on Mining Software Repositories.* 77–80. https://doi.org/10.1109/MSR.2013.6624010

[27] D I.K. Sjoberg, A Yamashita, B C.D. Anda, A Mockus, and T Dyba. 2013. Quantifying the Effect of Code Smells on Maintenance Effort. *IEEE Transactions on Software Engineering* 39, 8 (8 2013), 1144–1156. https://doi.org/10.1109/TSE.2012.89

[28] Z Soh, A Yamashita, F Khomh, and Y.-G. Gueheneuc. 2016. Do Code Smells Impact the Effort of Different Maintenance Programming Activities?. In *23rd International Conference on Software Analysis, Evolution, and Reengineering.* IEEE, 393–402.

[29] A Strauss and J Corbin. 1998. *Basics of Qualitative Research: Techniques and Procedures for Developing Grounded Theory.* SAGE Publications.

[30] S Subramanian, L Inozemtseva, and R Holmes. 2014. Live API Documentation. In *36th International Conference on Software Engineering.* ACM, New York, NY, USA, 643–652. https://doi.org/10.1145/2568225.2568313

[31] A Tahir, S Counsell, and S G MacDonell. 2016. An Empirical Study into the Relationship Between Class Features and Test Smells. In *2016 23rd Asia-Pacific Software Engineering Conference (APSEC).* IEEE, 137–144.

[32] A Tahir, A Yamashita, S Licorish, J Dietrich, and S Counsell. 2018. Supplementary material for "can you tell me if it smells? A study on how developers discuss code smells and anti-patterns in Stack Overflow". Dataset. (2018). https://doi.org/10.5281/zenodo.1241245

[33] N Tsantalis and A Chatzigeorgiou. 2009. Identification of Move Method Refactoring Opportunities. *IEEE Transactions on Software Engineering* 35, 3 (5 2009), 347–367. https://doi.org/10.1109/TSE.2009.1

[34] M Tufano, F Palomba, G Bavota, R Oliveto, M Di Penta, A De Lucia, and D Poshyvanyk. 2015. When and Why Your Code Starts to Smell Bad. In *37th International Conference on Software Engineering.* IEEE, Piscataway, NJ, USA, 403–414.

[35] S Wang, D Lo, and L Jiang. 2013. An empirical study on developer interactions in StackOverflow. In *28th Annual ACM Symposium on Applied Computing.* ACM Press, New York, New York, USA, 1019. https://doi.org/10.1145/2480362.2480557

[36] A Yamashita. 2014. Assessing the capability of code smells to explain maintenance problems: an empirical study combining quantitative and qualitative data. *Empirical Software Engineering* 19, 4 (8 2014), 1111–1143. https://doi.org/10.1007/s10664-013-9250-3

[37] A Yamashita and S Counsell. 2013. Code smells as system-level indicators of maintainability: An Empirical Study. *Journal of Systems and Software* (2013).

[38] A Yamashita and L Moonen. 2013. Do developers care about code smells? An exploratory survey. In *20th Working Conference on Reverse Engineering.* IEEE, 242–251.

[39] M Zhang, T Hall, and N Baddoo. 2011. Code Bad Smells: a review of current knowledge. *Journal of Software Maintenance and Evolution: Research and Practice* 23, 3 (4 2011), 179–202. https://doi.org/10.1002/smr.521