# A large-scale empirical study of practitioners' use of object-oriented concepts

Tony Gorschek
Blekinge Institute of Technology
Ronneby Sweden
tony.gorschek@bth.se

Ewan Tempero
Auckland University
Auckland, New Zealand
e.tempero@cs.auckland.ac.nz

Lefteris Angelis
Aristotle University of Thessaloniki
Thessaloniki, Greece
lef@csd.auth.gr

## ABSTRACT

We present the first results from a survey carried out over the second quarter of 2009 examining how theories in object-oriented design are understood and used by software developers. We collected 3785 responses from software developers world-wide, which we believe is the largest survey of its kind. We targeted the use of encapsulation, class size as measured by number of methods, and depth of a class in the inheritance hierarchy. We found that, while overall practitioners followed advice on encapsulation, there was some variation of adherence to it. For class size and depth there was substantially less agreement with expert advice. In addition, inconsistencies were found within the use and perception of object-oriented concepts within the investigated group of developers. The results of this survey has deep reaching consequences for both practitioners and researchers as they highlight and confirm central issues.

## Categories and Subject Descriptors

D.2.2 [**Design Tools and Techniques**]: Object-oriented design methods

## General Terms

Survey

## Keywords

Encapsulation, Number of methods, Inheritance depth

## 1. INTRODUCTION

In engineering, the usefulness of a theory is determined by whether it has a practical application, and whether it has a practical application is first determined by whether it is actually used, and second, whether it has been used properly. For engineers, the real world application of theory is one type of validation [12]. There are many theories for software engineering, and in particular for object-oriented design. We would like to understand what theories of object-oriented design are being used in practice.

Not all used theories are necessarily good ones however. An engineer may use a theory unaware that a better alternative exists. Conversely, a engineer may not use a theory due to doubts as to its usefulness. Thus, to evaluate the practical usefulness of a theory it is not enough to know that it is used, it is also important to understand *why* it has been used, and *how* it is used. To gain this understanding, we must look beyond the theories; we must talk to the engineers. In this paper we present the first results from a survey carried out over the second quarter of 2009 examining how theories in object-oriented design are understood and used by software developers. We collected 3785 responses from software developers world-wide, which we believe is the largest survey of its kind.

There are three main goals of the our empirical study, of which parts are presented in this paper. The first is to gauge how the theory and best-practice put forward by experts is used and interpreted by practitioners. Second is to identify the actual preferences of practitioners when it comes to practical issues such as making design decisions, how they interpret and understand the central concepts, and how they utilize the strengths of those concepts. Third, but probably most important, we want to determine whether or not the practitioners are in agreement. That is, to what degree do the practitioners, regarding how they work, what they prefer, and what they consider to be important, vary. This third part speaks to the ability of good practice, as suggested by experts, to penetrate the every day work performed by practitioners. In addition, if practitioners agree on what is good design practice, and if this differs from the good practice put forward by experts, researchers need to take a hard look at what this entails for state-of-the-art in research. If practitioners disagree on a large scale however, the implications could be even greater.

Much advice has been and is given regarding how software should be designed at many granularities, such as criteria for modules [17], common design idioms [10], and use of components [23]. With the increase in availability of open-source software, a number of empirical studies have been performed in recent years that give a picture of how software is really being written, and the results do not always appear consistent with received wisdom (see section 2). We would like to know whether the perceived inconsistency is real, whether it is due to lack of understanding by developers, lack of belief that the design principles have the claimed benefits, or whether the design principles in fact do not have the claimed benefits. As the bulk of the empirical studies is on code written in object-oriented languages, in our survey we focus on principles for object-oriented design.

The paper is organised as follows. Section 2 presents the relevant background and related work. Section 3 presents the research methodology, study design and operation, and Section 4 presents our results. Section 5 discusses our results and finally we present our conclusions in Section 6.

## 2. BACKGROUND AND RELATED WORK

The three concepts we examine are: encapsulation, class size as measured by number of methods, and depth of class in the inheritance hierarchy. We chose these concepts because they are simple enough to reduce misinterpretation in a survey context. In this section, we discuss what advice is offered regarding these concepts, and what other research has examined in relation to how the concepts are actually applied.

The advice is, roughly, "always encapsulate", don't have "too many" methods, and don't have classes "too deep" in the hierarchy, however the consistency of this advice is variable. Of the three, encapsulation has the strongest message, which is "always encapsulate." Others have noted that what encapsulation means has changed from its original meaning [2]. In this work, we use the meaning of "hiding representation".

Parnas proposed that the criteria for creating modules was to determine what decisions are likely to change, and hide those decisions within modules [17]. As the representation of an object is a decision that is considered to be likely to change, usually Parnas' criteria has been interpreted as that representation should be hidden. That is, the fields (or instance variables or attributes) should be inaccessible from outside the class, i.e. "encapsulated".

Some languages directly enforce this advice. Smalltalk, for example, does not allow any access to fields from clients that instantiate classes, although, as Schärli et al. note, clients that inherit from classes do have access [21]. They observe that this meant that inheriting classes were not protected from changes to representation and proposed a means to reduce this exposure. Inherent in their advice is the belief that all fields should be inaccessible from all other classes.

The advice regarding hiding representation is generally consistent. Riel's first heuristic for object-oriented design is "All data should be hidden within its class"[19] (Heuristic 2.1), introduction to programming books often give similar advice (for example Sedgewick and Wayne "insist" on the convention of making all fields private [22], p422), Fowler lists "Inappropriate Intimacy" as a bad smell [9], Lorenz and Kidd comment that they "... do not believe in public instance variables, since that breaks encapsulation" [16].

Encapsulation of fields is also implied by some metrics. For example, the "coupling between objects" (CBO) metric proposed by Chidamber and Kemerer counts accesses through fields as well as methods, implying such accesses are to be avoided [6]. Fields that are not accessible (private) could therefore not contribute to measurements by this metric. The MOOD metric suite includes the metric "attribute hiding factor", of which the authors say "Ideally, all attributes would be hidden and only accessed by the corresponding class methods" [4].

There are any number of websites who claim good object-oriented design requires fields be encapsulated. While websites are not the most authoritative sources of advice, in this case they do generally agree with prominent researchers, books on good design, and programming texts, and are also perhaps more likely to be accessible to practitioners than the research literature.

A commonly-suggested alternative to public fields is to provide *accessor* methods — methods that exist to either return the value of a field ("getters") or change the value of a field ("setters"). This satisfies the letter of rules such as "all data should be hidden". Some languages, such as C# and Eiffel, provide syntactic sugar for accessors, often referred to as *properties*. The intent is to provide protection from arbitrary access to the fields while having the convenience of simpler syntax. However, as has been observed, if used carelessly accessor methods will still expose implementation decisions such as the type of the field, and claim good object-oriented design should not need to provide any access to fields [14].

Despite the advice regarding encapsulation being fairly consistent, there are nevertheless apparently authoritative examples of it not being followed. For example, the Java standard library contains several, such as `x` and `y` in the `java.awt.Point` class, and `in` and `out` in the `java.lang.System` class. The existence of such visible examples may encourage programmers, particularly novices, to ignore the advice. There is certainly evidence that the advice is not universality followed. In a recent study of 100 open-source Java applications, ranging in size from 42 to 17621 classes, 59% of the classes of half the applications had non-private fields, several applications had more than 20% of classes with public fields [24].

The advice regarding the number of methods in a class is less direct, nevertheless the general sense seems clear. The intuition is that if one thing is bigger than a second, then any action (especially cognitive action such as making a change) one wants to perform on the first is going to be more difficult in some sense than on the second. This intuition is the basis for one of the metrics proposed by Chidamber and Kemerer, WMC [6]. They argue the number of methods and their complexity "is a predictor of how much time and effort is required to develop and maintain the class". They cite Bunge's definition of complexity to justify their definition of WMC, however Bunge notes that the number of parts is a "coarse measure of ontic (nonconceptual) complexity" [5], implying that there are aspects of complexity that are not represented by counting parts.

There has been the suggestion that the above intuition is not correct, at least with respect to fault proneness, i.e. that there is a size such that modules either smaller or larger than that size are more fault prone (by Hatton for example, [13]). However, more recently the evidence suggests that this so-called "Goldilock's conjecture" is not true [7, 8]. The conclusion is then that one should avoid creating classes that are "too big". There is the question, however, of how is the size of a class measured and what exactly constitutes "too big"?

Fowler lists "Large Class", "too many instance variables" [9] and suggests "Extract Class" as a refactoring technique to reduce the number of fields, which is also likely to reduce the number of methods. He also suggests that classes with "too many responsibilities" (p141) or "too much behaviour" (p142) are candidates for Extract Class. Other techniques add methods such as "Inline class" however it is intended to be applied to a "class that isn't doing very much".

There is, however, some explicit advice limiting the number of methods. Lorenz and Kidd [16] (pp50–51) suggest a heuristic of threshold of 20 methods for "model" classes 40 for "UI" classes "thresholds", with averages of 12 and 25 respectively. They suggest thresholds of 3 for fields in model classes and 9 for UI classes.

Empirical studies appear not to support the use of hard limits on the number of methods in a class. Baxter et al. examined 56 open-source Java applications and reported distributions that are close to powerlaws (so called "truncated curves"), meaning that generally the larger the application, the more methods the largest class tends to have [1]. They indicated that large applications generally had classes with hundreds of methods.

Advice regarding depth in inheritance trees also varies. Johnson and Foote recommend that "Class hierarchies should be deep and narrow" [15] (Rule 5) whereas Booch suggests that developers tend to have classes no deeper than 7±2 [3] (p280). Riel perhaps best typifies this confusion, at the same time advising both — "In theory, inheritance should be deep — the deeper, the better" (Heuristic 5.4) and "In practice, inheritance hierarchies should be no deeper than an average person can keep in his or her short-term memory. A popular value for this depth is six" (Heuristic 5.5).

The empirical evidence regarding the size of inheritance hierarchies is not so clear. Tempero et al. studied 93 open-source Java applications and saw either powerlaw or truncated curve distributions for both depth and width (number of children) of classes [25]. However, while the shapes of the distributions suggested that the larger the application the greater the depth of the deepest class, none of the applications in the studied (which included large applications such as Eclipse) had classes at depth greater than 10.

The evidence from empirical studies is that the advice offered regarding encapsulation, number of methods, and class depth may fall on deaf ears. However, as much of the evidence comes from analysing source code, what it cannot tell us is anything about *intent*. It may be that by-and-large practitioners are aware of the advice, and are trying to follow it, but are failing due to misunderstanding or some other reason. Practitioners may also be aware of it but don't believe the advice to be relevant. Another possibility, which is even more worrying is that practitioners are simply not aware of it, implying failure in education and the spreading of research results to practice. Without knowing *why* practitioners make the decisions they do, we cannot fully evaluate the usefulness of the advice being offered. The only way we can find this out is to ask them.

## 3. RESEARCH METHODOLOGY

This section introduces the research questions, the study design and execution, and threats to validity.

### 3.1 Research Question

In order to investigate how object-oriented concepts are understood, and to what extent theory is practised, the following main research questions were formulated.

**RQ1:** How are object-oriented concepts *understood* and *used* in practice, and how does the use *compare* to recommended best-practice?

Based on this general RQ two sub-questions were formulated addressing the specific concepts of encapsulation, class size as measured by number of methods, and depth of a class in the inheritance hierarchy.

**RQ1.1:** How is the concept of encapsulation *understood* and *used* in practice, and how does the use *compare* to recommended best-practice?

**RQ1.2:** How are the concepts of size and depth *understood* and *used* in practice, and how does the use *compare* to recommended best-practice?

In this context we use "theory", "best-practice", and "advice" interchangeably.

### 3.2 Study Design and Operation

The survey was executed through the creation of a on-line questionnaire that was designed using a mix of closed and open ended questions [20]. The first version of the questionnaire was piloted using research programmers for the Computer Science department at the University of Auckland, New Zealand. As the test group took the survey, we monitored time, logged questions, and caught mis-understandings due to question formulation. We then followed with a debrief session. Based on the pilot, the survey instrument was improved. A second test of the instrument was performed using research colleagues at Blekinge Institute of Technology where four researchers (engineering and science PhDs) gave additional feedback on the instrument.

The survey had three parts. Part 1 gathered demographic information. Part 2 mainly addressed the concept of encapsulation (RQ1.1). Part 3 covered class size and class depth (RQ1.2). Part 1 had 7 questions with a mixture of question types, including single-choice (radio buttons) multiple-choice, multi-choice (checkboxes) multiple-choice, and free-text. All of the questions in parts 2 and 3 were either single-choice multiple choice or free-text, which all but one multiple-choice question having a free-text area allowing respondents to expand or qualify their answer. There were 5 questions in part 2 and 10 in part 3 for a total of 22.

The motivation for using an on-line questionnaire was to maximize coverage and participation. Surveys are an appropriate strategy for collecting empirical results from a large population, and given an adequate response rate, an understanding of the population can be achieved [18]. The questionnaire was made accessible on-line at `surveymonkey.com` between March and June of 2009.

Participants were recruited primarily through personal contacts and forums targeted at software developers, and encouraging those who participated to spread the word. We provided information about the goals of the survey on our website (`sefolklore.com`), and posted a video to YouTube. The idea behind our information campaign was to get a "snow ball effect", where "word-of-keyboard" spread information regarding the survey on the Internet. The campaign was successful in that the survey was eventually mentioned on twitter by a high-profile user, resulting in a very good response.

The theoretical population [11] for the study was any and all software developers with experience in either closed or open source development with experience in using any object-oriented programming language. The actual population, or sampling frame, was of course limited by Internet access and our ability to reach the developers within

| Continent | # respondents | percent |
|---|---|---|
| Africa | 1 | 0.03% |
| Asia | 294 | 7.77% |
| Australasia | 426 | 11.25% |
| Europe | 1207 | 31.89% |
| North America | 1724 | 45.55% |
| South America | 77 | 2.03% |

**Table 1: Geographical distribution of respondents.**

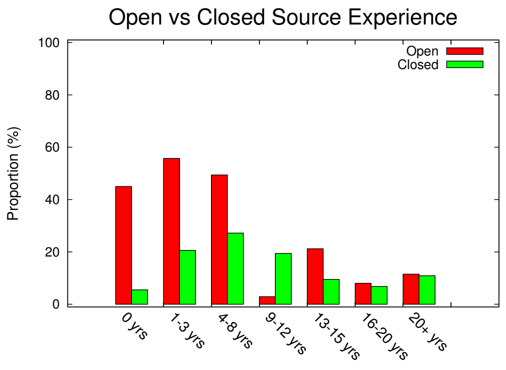

**Figure 1: Distribution of respondents based on years of development experience.**

the given timeframe (and their willingness to participate). From one aspect the sample can be described as convenience sampling [20] as we utilized primarily our own contacts initially, however the sample quickly spread beyond our sphere of influence and contacts, with several hundred respondents before the twitter post (approximately 10 days after the survey went live), and thousands after it.

A total of 4823 respondents started the survey, and 3785 completed all the mandatory questions, a completion rate of 78.5%, which is well above expected given that the survey was substantial (three full pages, demanding about 15-20 minutes, and judging by the free-text responses a large number may have spent even more).

## 3.3 Validity Evaluation

We consider the four perspectives of validity and threats as presented in Wohlin et al. [26].

### 3.3.1 *Construct validity*

The construct validity is concerned with the relation between the theories behind the research and the observations. The variables in our research are measured through the survey, including closed as well as open-ended questions where the participants are asked to share their professional experiences as developers. Mono-operation bias was avoided by collecting data from a wide range of sources on the topic of the study, and the theory was richly elaborated through the questionnaire by posing multiple questions on the same topic. To avoid evaluation apprehension, complete anonymity of the subjects was guaranteed. There is always a risk that the background of the subjects (e.g. experience) is a central influence, however, due to the large sample, as well as the spread of competence and level of experience we feel the risk is limited. Hypothesis guessing (the respondents try to guess what the researchers want) is also a potential threat. The introduction to the survey (video and web page) stressed the importance of honesty, however this threat can not be completely dismissed.

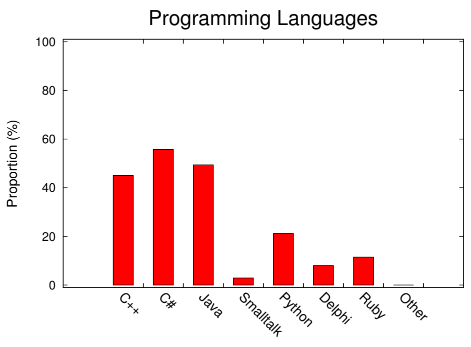

**Figure 2: Distribution of respondents based on language proficiency.**

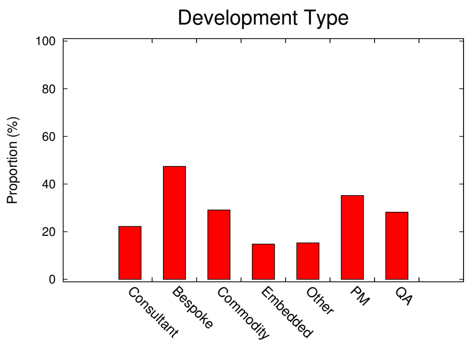

**Figure 3: Distribution of respondents based on type of development.**

### 3.3.2 *Conclusion validity*

Threats to conclusion validity are concerned with the possibility of incorrect conclusions about a relationship in observations that may arise from error sources such as, instrumental flaws, influence posed on the subjects, or selection. We can not exclude the possibility that the instrumentation (survey questions, formulations, explanations etc) were misunderstood by the subjects, however, pre-tests and reviews by colleagues, as well as redundancy (several questions addressing the same thing), hopefully alleviated the risks of this threat. Regarding subject influence, there can be a chance that some subjects interacted (e.g. colleagues at a work place) and that this interaction influenced some of the subjects' answers. However, due to the response rate, as well as substantial sample we feel that the overall influence of this is limited. The sample selected for the study were developers, however, we feel that the group was fairly heterogeneous (experience, education etc). In a small sample this might influence the outcome, however, due to sample size the risk of differences between subject unduly influencing the result is low.

### 3.3.3 *Internal validity*

This threat is related to issues that may affect the causal relationship between treatment and outcome. Threats to internal validity include instrumentation, maturation and

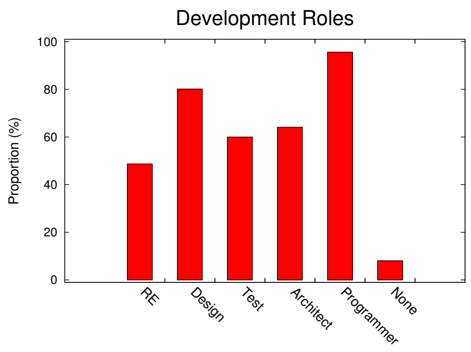

**Figure 4: Distribution of respondents based on development roles.**

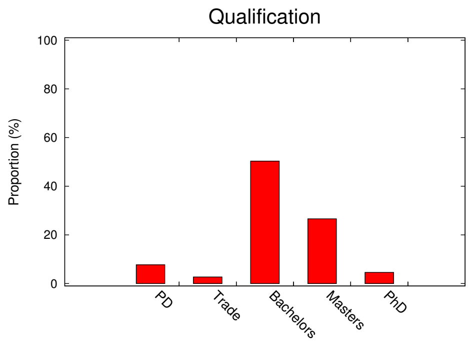

**Figure 5: Distribution of respondents based on qualification.**

selection threats. In our study, the instrument was pretested, as mentioned before. Maturation pertains to, for example, learning effect or subject's responses being influenced by boredom. Each subject participated once, thus learning effect was small, and the questionnaire took about 20 minutes to complete. In addition, the high completion rate of respondents (a clear majority of the respondents who started the survey also finished it) indicates an interest in participating, indicating that the respondents took an interest in being thorough in their efforts to answer the questions. The interest of the subject may influence the representativeness of the subjects. This is a hard threat to counter as willingness to participate and interest in the subject are associated. The large sample may alleviate this to a degree, however the threat can not be dismissed. Further, the selection of subject was performed by using a wide range of media and channels, far beyond the control or sphere of influence of the researchers.

### 3.3.4 *External validity*

This threat is concerned with the ability to generalize the findings beyond the actual study. The actual setting of the study was an environment known to the subjects (from home/office using the web), thus our control and influence of the context was minimal. In addition, the sample was very similar to the population, that is, developers with experience in object-oriented programming. In addition, due to sample size we feel confident that generalizability of the results are possible.

**Q8.** Which of the following best describes your thinking when adding a field to a class?
**1.** I always make it private (inaccessible outside the class)
**2.** I usually make it private
**3.** I never make it private,
**4.** I don't think about it, I pick whatever is convenient
**5.** I don't know what you mean/None of the options describe my thinking)

**Q9.** Which of the following best describes your thinking when providing access to fields?
**1.** I always provide "getters" and "setters"
**2.** I avoid "getters" and "setters" wherever possible
**3.** I prefer to avoid "getters" and "setters" but will provide them rather than make large changes to avoid them,
**4.** I don't think about it,
**5.** I don't know what you mean

**Figure 6: Text of Questions 8 & 9**

## 4. RESULTS AND ANALYSIS

This section presents the results of the survey, organised according to the research questions presented in Section 3. This paper presents only a portion of the data gathered due to space constraints. We focus on the survey questions directly relevant to RQ1.1 and RQ1.2.

Not all of the questions were compulsory and so in reporting completed surveys we refer to those where all compulsory questions were answered. We will only give full details here of those questions we report on; the full survey is available on `sefolklore.com`.

### 4.1 Respondent Demographics

A total of 4823 respondents initiated the survey, and 3785 completed the compulsory questions of the survey. Responses came from 84 different countries. The distribution of re-

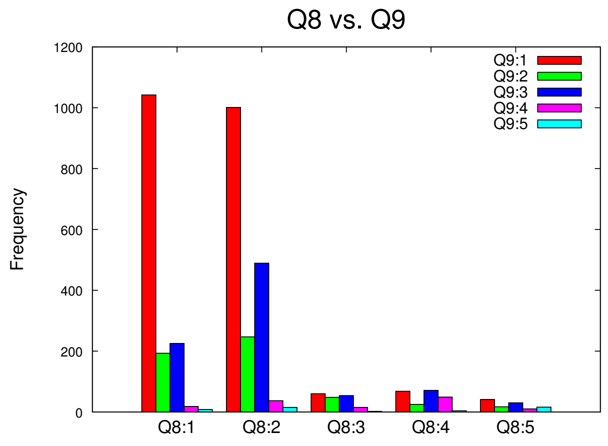

**Figure 7: Q8 vs. Q9**

| | Q9 | | | | | |
|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 | Total |
| **Q8:1** | 1042 | 193 | 225 | 18 | 8 | 1486 |
| % Q8 | 70.1 | 13.0 | 15.1 | 1.2 | 0.5 | 100.0 |
| % Q9 | 47.1 | 36.4 | 25.9 | 14.0 | 17.8 | 39.3 |
| % Total | 27.5 | 5.1 | 5.9 | 0.5 | 0.2 | 39.3 |
| **Q8:2** | 1001 | 247 | 489 | 37 | 15 | 1789 |
| % Q8 | 56.0 | 13.8 | 27.3 | 2.1 | 0.8 | 100.0 |
| % Q9 | 45.3 | 46.6 | 56.3 | 28.7 | 33.3 | 47.3 |
| % Total | 26.4 | 6.5 | 12.9 | 1.0 | 0.4 | 47.3 |
| **Q8:3** | 60 | 48 | 54 | 15 | 2 | 179 |
| % Q8 | 33.5 | 26.8 | 30.2 | 8.4 | 1.1 | 100.0 |
| % Q9 | 2.7 | 9.1 | 6.2 | 11.6 | 4.4 | 4.7 |
| % Total | 1.6 | 1.3 | 1.4 | 0.4 | 0.1 | 4.7 |
| **Q8:4** | 68 | 25 | 71 | 49 | 4 | 217 |
| % Q8 | 31.3 | 11.5 | 32.7 | 22.6 | 1.8 | 100.0 |
| % Q9 | 3.1 | 4.7 | 8.2 | 38.0 | 8.9 | 5.7 |
| % Total | 1.8 | 0.7 | 1.9 | 1.3 | 0.1 | 5.7 |
| **Q8:5** | 41 | 17 | 30 | 10 | 16 | 114 |
| % Q8 | 36.0 | 14.9 | 26.3 | 8.8 | 14.0 | 100.0 |
| % Q9 | 1.9 | 3.2 | 3.5 | 7.8 | 35.6 | 3.0 |
| % Total | 1.1 | 0.4 | 0.8 | 0.3 | 0.4 | 3.0 |
| **Total** | 2212 | 530 | 869 | 129 | 45 | 3785 |
| % Q8 | 58.4 | 14.0 | 23.0 | 3.4 | 1.2 | 100.0 |
| % Q9 | 100.0 | 100.0 | 100.0 | 100.0 | 100.0 | 100.0 |
| % Total | 58.4 | 14.0 | 23.0 | 3.4 | 1.2 | 100.0 |

**Table 2: Q8 vs. Q9 Cross tabulation.**

sponses by continent is shown in Table 1. As we can seen a clear majority of the responses originate from North America with Europe following closely.

Just over half the respondents had some open-source development experience, with the half of those having done 1-3 years of open source development. Almost all (94.5%) had done closed-source development, with approximately half having done 1-8 years and 10% having done more than 20 years (see Figure 1).

For languages used (Figure 2), 91% were experienced with one of C# (56%), Java (49%), or C++ (45%). The fourth most chosen language, Python (21%) adds another 2.8%, meaning there were 235 (6.2%) of respondents who were not experience in any of these languages. Of those, almost all (191 or 81%) indicated experience in a language not listed explicitly in the question.

About half (47.4%) of the respondents had experience in bespoke software development (Figure 3) and almost all (95.6%) claimed experience in programming (Figure 4).

With respect to the respondents level of academic qualifications (Figure 5), 2420 (50.2%) had a Bachelors degree, and 1245 (25.8%) had a Masters degree. About 10% also had participated in trade certificates/professional development courses. A total of 429 (8.9%) indicated having no formal training or degree.

## 4.2 RQ1.1 Encapsulation

To gauge how encapsulation is understood and used in practice we consider Question 8 (Q8) and Question 9 (Q9) as shown in Figure 6. These two questions were intended to capture practitioners' attitude towards encapsulation, albeit without using that term. From our discussion in Section 2, if practitioners follow the expert advice then they would make fields private (Q8, alternative 1). Whether they would routinely provide getters and setters is not so clear.

The chart in Figure 7 shows one view of how survey participants answered these two questions. We use the notation shown to indicate alternatives answered for questions, for example "Q8:1" denotes Question 8, alternative 1. Table 2 shows the same data (in the "Total" row and column) as well as cross-tabulating the responses. The "% Q8" and the "% Q9" row indicates the *conditional* probability. For example, in the row "% Q8" we see 1042/1486 = 0.701 = 70.1%, implying that the there is a probability of 70.1% that, for respondents that have answered Q8:1, they will also have answered Q9:1. The row "% Q9" indicates the same for Q9, that is for example, there is a probability of 47.1% that the ones who have already answered Q9:1 will also have answered Q8:1. The column "% Total" indicates how many respondents answered Q8:1 and Q9:1, that is, the probability of intersection P(Q8:1 and Q9:1)=0.275 (27.5%).

Based on the results shown in Table 2 we cannot claim that the advice regarding encapsulation, at least as interpreted as representation hiding, is being followed without question. Many, but not quite half (1486/3785, 45.4%) indicated they might consider no private fields. A more generous interpretation of this question might consider Q8:1 and Q8:2 to indicate adherence to expert advice, in which case, 86.5% of respondents could be said to follow the advice — but about 13.5% do not. The responses to Q9 are much more clear-cut — 58.4% indicating they would provide accessors, and if Q9:2 were included then 72.4% could be said to follow this advice.

For those who do make their fields private (Q8:1), they are more likely than not (70.1%) to provide accessors (Q9:1). Expanding this to Q8:2 and Q9:2, then 75.8% routinely provide accessors to private fields. For those who provide getters and setters (Q9:1, Q9:2), 90.6% of them (Q8:1, Q8:2) will make their fields private. This is perhaps not surprising, as there is not much point having getters and setters if fields are public. However it is also the case that 50.8% of those who generally do not make fields private (Q8:3, Q8:4) also provide getters and setters! This suggests some confusion as to the role of these kinds of methods.

Performing a chi-square test [26] on the frequencies gives a value of significant of $p < 0.0005$, showing a significant association between the answers of Q8 and Q9 (a value <0.05 shows significant dependence (association) between the responses of each pair tested). The actual cause of this dependency is not totally clear and demands further analysis of the open question text answers given in association to the question. However, a preliminary possibility could be that the respondents had a specific wish to adhere to good practices (enabling encapsulation), but do not consider the use of getters and setter to be inconsistent with good encapsulation.

## 4.3 RQ1.2 Size and Depth

For RQ1.2 we are trying to determine to what degree class size (as measured by number of methods) or class depth figured as design decisions. Figure 8 shows two questions relating to class size and the possible responses. Table 3 shows the responses to these questions. About 10.9% (Q13:1) cared enough about class size to specifically track it, however nearly 50% (Q13:2) were at least aware of the number of methods. Almost 39% (Q13:3-4) didn't really care about

**Q13.** When working on a class, how aware are you of the number of methods it has?
**1.** I always try to determine how many methods a class has
**2.** I will note classes that seem to have a lot of methods
**3.** The number of methods is usually not relevant for me
**4.** I don't think about it
**5.** I don't know what you mean

**Q14.** What do you think is the largest number of methods a class should have? (Choose one of the alternatives, and add a number in the textbox that replaces "N" in the text of the alternative you chose)
**1.** There should be never more than about N methods.
**2.** I try to avoid having more than about N methods in a class, but will allow exceptions in extreme circumstances.
**3.** I prefer to avoid having more than N methods in a class but I am not fanatical about it.
**4.** I don't really think about how many methods there are in a class but prefer to avoid having classes with more than N methods.
**5.** I don't think there should be any limit on the number of methods in a class.

**Figure 8: Text of Questions 13 & 14**

the number of methods in a class.

Looking at the answers to Q14, we can see that the respondents choosing alternatives 1-3 (42.3%) are for a limit of some sort with regard to number of methods, while the respondents choosing alternative 4 and 5, didn't care or were against a limit (57.8%). This would indicate that the field is split, half preferring a limit, and half not in favour of one.

Table 3 gives an overview of Q13 and Q14 in combination. Performing a chi-square test [26] on the frequencies gives a value of significant $p < 0.0005$, showing a significant association between the answers of Q13 and Q14. Comparing alternatives 1 and 2 for Q13 and alternatives 4 and 5 for Q14, there are 934 (24.7%) respondents who chose one of the 4 possible pairs, so nearly one quarter of respondents reported, on the one hand, being aware of how many methods a class had, but on the other hand did not act on that information. That said, the combination with the largest number of response was Q13:2 and Q14:3, with 839 (22.2%) responses, which suggests at least an awareness that classes should not have "too many" methods.

Q14 also requested that respondents suggest a maximum number of methods they would expect for a class. The results for this are shown in Tables 4 and 5. From Table 4, of the 45 respondents who thought there should be an absolute limit (Q14:1), the median was 10, with the largest suggested limit being 400, and the mode was 10, indicating that the respondents of Q14:1 preferred a moderate class size. Of those who were somewhat supportive of a limit (responses 1-4, 2450 responses — not shown in the table), the median was 15 and maximum was 1000 (in fact only 4 suggested larger than 100 – see Table 5), and mode 10. Almost all (93.6%) of those who were supported of a limit and suggested a value of 30 methods or less. While this range is not great, it is wide enough to indicate a general lack of agreement regarding appropriate class size. However, a clear majority of the ones being for a size limitation were moderate in the size recommendation.

| | **Q14** | | | | | |
|---|---|---|---|---|---|---|
| | **1** | **2** | **3** | **4** | **5** | **Total** |
| **Q13:1** | 19 | 133 | 128 | 54 | 77 | 411 |
| % Q13 | 4.6 | 32.4 | 31.1 | 13.1 | 18.7 | 100.0 |
| % Q14 | 42.2 | 32.7 | 11.2 | 6.3 | 5.8 | 10.9 |
| % Total | 0.5 | 3.5 | 3.4 | 1.4 | 2.0 | 10.9 |
| **Q13:2** | 18 | 227 | 839 | 391 | 412 | 1887 |
| % Q13 | 1.0 | 12.0 | 44.5 | 20.7 | 21.8 | 100.0 |
| % Q14 | 40.0 | 55.8 | 73.1 | 45.9 | 30.9 | 49.9 |
| % Total | 0.5 | 6.0 | 22.2 | 10.3 | 10.9 | 49.9 |
| **Q13:3** | 5 | 36 | 156 | 329 | 687 | 1213 |
| % Q13 | 0.4 | 3.0 | 12.9 | 27.1 | 56.6 | 100.0 |
| % Q14 | 11.1 | 8.8 | 13.6 | 38.7 | 51.5 | 32.0 |
| % Total | 0.1 | 1.0 | 4.1 | 8.7 | 18.2 | 32.0 |
| **Q13:4** | 3 | 9 | 23 | 75 | 152 | 262 |
| % Q13 | 1.1 | 3.4 | 8.8 | 28.6 | 58.0 | 100.0 |
| % Q14 | 6.7 | 2.2 | 2.0 | 8.8 | 11.4 | 6.9 |
| % Total | 0.1 | 0.2 | 0.6 | 2.0 | 4.0 | 6.9 |
| **Q13:5** | 0 | 2 | 1 | 2 | 7 | 12 |
| % Q13 | 0.0 | 16.7 | 8.3 | 16.7 | 58.3 | 100.0 |
| % Q14 | 0.0 | 0.5 | 0.1 | 0.2 | 0.5 | 0.3 |
| % Total | 0.0 | 0.1 | 0.0 | 0.1 | 0.2 | 0.3 |
| **Total** | 45 | 407 | 1147 | 851 | 1335 | 3785 |
| % Q13 | 1.2 | 10.8 | 30.3 | 22.5 | 35.3 | 100.0 |
| % Q14 | 100.0 | 100.0 | 100.0 | 100.0 | 100.0 | 100.0 |
| % Total | 1.2 | 10.8 | 30.3 | 22.5 | 35.3 | 100.0 |

**Table 3: Q13 vs. Q14 Cross tabulation.**

| Q14 | Freq. | Ave | Median | Mode | Max. | Std. Dev. |
|---|---|---|---|---|---|---|
| Q14:1 | 38 | 23.29 | 10.00 | 10 | 400 | 63.404 |
| Q14:2 | 376 | 14.69 | 10.00 | 10 | 100 | 10.649 |
| Q14:3 | 1086 | 15.71 | 12.00 | 10 | 200 | 12.315 |
| Q14:4 | 781 | 23.24 | 20.00 | 20 | 1000 | 42.807 |
| Q14:5 | 70 | 797.86 | 12.00 | 0 | 10000 | 2530.824 |
| Total | 2351 | 41.46 | 15.00 | 10 | 10000 | 454.315 |

**Table 4: Descriptive statistics for choice of N (maximum number of methods) for Q14.**

Figure 9 shows two questions relating to class depth and the possible responses. Table 8 shows the cross-tabulation of the results. About 897 (23.7%) of the respondents always determine class depth (Q17:1), 2101 respondents have a rough idea (Q17:2), but a substantial 755 (19.9%) of the respondents don't care at all (Q17:3). For Q18 respondents choosing alternatives 1, 2, and 3 (51.3%) are for being aware of class depth, and limiting the depth. Respondents answering alternatives 4 and 5 of Q18, a total of 1846 respondents (48.7%) either don't think about it, or do not think there should be any limitation at all.

Table 8 shows the comparison of Q17 and Q18 (a chi-square test showed a value of $p < 0.0005$). The higher proportions of responses for Q17 alternatives 1 and 2 suggests a higher awareness and concern for the depth of classes

| Range | Frequency |
|---|---|
| None | 169 |
| 0–10 | 951 |
| 11–20 | 975 |
| 21–30 | 209 |
| 31–40 | 24 |
| 41–50 | 80 |
| 51–60 | 2 |
| 61–70 | 0 |
| 71–80 | 4 |
| 81–90 | 0 |
| 91–100 | 32 |
| 100+ | 4 |

**Table 5: Distribution of choices of N for limit on number of methods by participants who chose alternatives 1-4 for Q13**

| Q18 | Freq. | Ave | Median | Mode | Max. | Std. Dev. |
|---|---|---|---|---|---|---|
| Q18:1 | 211 | 3.43 | 3.00 | 3 | 10 | 1.820 |
| Q18:2 | 686 | 3.72 | 3.00 | 3 | 12 | 1.512 |
| Q18:3 | 939 | 5.37 | 4.00 | 3 | 999 | 32.817 |
| Q18:4 | 691 | 5.79 | 5.00 | 5 | 100 | 5.301 |
| Q18:5 | 52 | 262.31 | 5.00 | 0 | 10000 | 1396.507 |
| Total | 2579 | 10.06 | 4.00 | 3 | 10000 | 200.728 |

**Table 6: Descriptive statistics for choice of N (maximum class depth) for Q18.**

compared with size of classes, as can be seen in comparing Q13 with Q17. 32% of the respondents in Q13 thought the number of methods (size) was irrelevant, while for Q17 only 19.9% considered the depth to be irrelevant. This is also reflected in the responses to Q18, which show increased responses for alternatives 1 and 2 compared to the equivalent alternatives for Q14. Nevertheless there were also an larger number of responses for alternatives 4 and 5 for Q18, particularly for those who gave alternatives 1 or 2 for Q17 than the equivalents for Q14 and Q13.

Looking at Q17 and Q18, overall, 1186 respondents (31%) chose one of the 4 pairs indicating an awareness of depth but a tendency not to act on this information (one of Q17:1 & Q17:2 and one of Q18:4 & Q18:5).

Tables 6 and 7 show some of the data regarding choices by participants of maximum depth of classes. Table 7 shows the choices of those who thought there should be a hard limit on the depth (Q18:1, 226 responses), the median was 3 and the maximum 10. Of those who were generally supportive of a limit (Q18:1-4, 2682 responses), the median was 4 and the maximum 999 (in fact there are only two proposed limits larger than 50, the most usual limit was between 3 and 5).

## 5. DISCUSSION

How are object-oriented concepts *understood* and *used* in practice, and how does the use *compare* to recommended best-practice?

Regarding RQ1.1, on how is the concept of encapsulation is *understood* and *used* in practice, and how does the use *compares* to recommended best-practice, there is general, but by no means universal, adherence to the advice of "hide representation". There is, however, significant variation among the respondents, in both their claimed use and their understanding of the principles involved.

As well as the multiple-choice questions, we also asked for free-text responses. We have not analysed these formally but some responses jump out. While we did not use the term "encapsulation", it was mentioned quite often in the free-text responses, for example (some seemingly representative statements), "One of the basic OO principles is data encapsulation" and "That's a golden rule: encapsulation is a key tenet of OO programming." There was also confusion evident in the use of accessors. Some associated their use with encapsulation: "I want to provide encapsulation for my classes, so I hide the fields using getter/setter pairs" but others did not: "Getters and setters break encapsulation." There were also some who were just not convinced: "encapsulation is mostly overrated."

Q17. When working on a class, is its depth in the inheritance hierarchy important to you?
**1.** I always try to determine how deep a class is
**2.** I usually have a rough idea of the depth of a class
**3.** It is not relevant for me
**4.** I don't know what you mean

Q18. What do you think the maximum depth of a class should be in the inheritance hierarchy? (Choose one of the alternatives, and add a number in the textbox that replaces "N" in the text of the alternative you chose)
**1.** No class should be deeper than N
**2.** I try to avoid having classes deeper than N but will allow it in extreme circumstances.
**3.** I prefer not to have classes deeper than N but I am not fanatical about it.
**4.** I don't really think about how deep classes are but probably would avoid having them deeper than N
**5.** I do not think there should be a limit to the depth of a class.

**Figure 9: Text of Questions 17 & 18.**

| N | Frequency |
|---|---|
| None | 15 |
| 0 | 4 |
| 1 | 16 |
| 2 | 38 |
| 3 | 78 |
| 4 | 27 |
| 5 | 30 |
| 6 | 4 |
| 7 | 6 |
| 8 | 2 |
| 9 | 2 |
| 10 | 4 |

**Table 7: Distribution of choices of N (maximum depth of classes) by participants who answered Q18:1.**

|  | Q18 |  |  |  |  |  |
|---|---|---|---|---|---|---|
|  | **1** | **2** | **3** | **4** | **5** | **Total** |
| **Q17:1** | 123 | 298 | 240 | 70 | 166 | 897 |
| % Q17 | 13.7 | 33.2 | 26.8 | 7.8 | 18.5 | 100.0 |
| % Q18 | 54.4 | 41.0 | 24.3 | 9.4 | 15.0 | 23.7 |
| % Total | 3.2 | 7.9 | 6.3 | 1.8 | 4.4 | 23.7 |
| **Q17:2** | 73 | 396 | 682 | 460 | 490 | 2101 |
| % Q17 | 3.5 | 18.8 | 32.5 | 21.9 | 23.3 | 100.0 |
| % Q18 | 32.3 | 54.5 | 69.1 | 61.9 | 44.4 | 55.5 |
| % Total | 1.9 | 10.5 | 18.0 | 12.2 | 12.9 | 55.5 |
| **Q17:3** | 29 | 32 | 63 | 204 | 427 | 755 |
| % Q17 | 3.8 | 4.2 | 8.3 | 27.0 | 56.6 | 100.0 |
| % Q18 | 12.8 | 4.4 | 6.4 | 27.5 | 38.7 | 19.9 |
| % Total | 0.8 | 0.8 | 1.7 | 5.4 | 11.3 | 19.9 |
| **Q17:4** | 1 | 0 | 2 | 9 | 20 | 32 |
| % Q17 | 3.1 | 0.0 | 6.2 | 28.1 | 62.5 | 100.0 |
| % Q18 | 0.4 | 0.0 | 0.2 | 1.2 | 1.8 | 0.8 |
| % Total | 0.0 | 0.0 | 0.1 | 0.2 | 0.5 | 0.8 |
| **Total** | 226 | 726 | 987 | 743 | 1103 | 3785 |
| % Q17 | 6.0 | 19.2 | 26.1 | 19.6 | 29.1 | 100.0 |
| & Q18 | 100.0 | 100.0 | 100.0 | 100.0 | 100.0 | 100.0 |
| % Total | 6.0 | 19.2 | 26.1 | 19.6 | 29.1 | 100.0 |

**Table 8: Q17 vs. Q18 Cross tabulation.**

Next, looking at size and depth of classes, the responses from the survey varied. About half of the respondents where for some sort of limit to class size, with most being of the opinion that 10 to 20 methods should be the maximum, and the other half didn't really care for any sort of limit. The implication being that about half of the respondents agree with the experts that there should be a limit, and the overall consensus of 10 to 20 methods seems to be well in the recommended range. On the other hand, the 22.5% of the respondents who didn't really care, and the 35.3% who were for no limit at all, can be said to be in disagreement with the accepted wisdom of the experts. The choices for the maximum number of methods was fairly consistent no matter what alternative was chosen for Q14 (including Q14:5!), being in the range 10–20. This is consistent with the advice given by Lorenz and Kidd [16], but not consistent with how code is actually being written [1].

The same division of the respondents into two groups can be seen when it comes to class depth. Even if most respondents had a rough idea of how deep a class they were working on was, when it came to implications the results varied. A total of 51.3% of the respondents were in line with the experts recommendations and opted for a limitation in class depth, most opting for a limit of 10. On the other hand, 48.7% of the respondents either didn't care, or where against any limits on class depth.

An interesting, but far from conclusive, indication could be seen when comparing views on size and depth. The results seem to indicate that class depth is considered more important to monitor (i.e. know the depth of a class) than the size of a class.

Thus, answering RQ1.2: How are the concepts of Size and Depth *understood* and *used* in practice, and how does the use *compare* to recommended best-practice? There does not seem to be a consensus, but rather a clear division between the respondents on what constitutes good practice when it comes to both class size and depth.

The data presented in this paper are not complete, and future work includes detailed analysis of motivation and free text qualifications of responses offered by many respondents, however some central and worrying tendencies can be observed. Disagreement among the respondents of what constitutes good utilization of object-oriented concepts bears possible serious implications. This is further aggravated by the fact that this also shows a gap between the commonly held wisdom put forward by researchers and experts and practitioners.

There can be several interpretations of the results from this survey. One interpretation can be that there is not a unified view among practitioners on what actually works in everyday development. For example, in many cases a limit in, for example, class depth or size might be counterproductive in the real world as experienced by the respondents. This interpretation would imply that the experts and researchers need to increase their efforts and perform more empirical investigations on both the artifacts (the software/systems), and the engineering practice and context of development in order to study reality.

An other interpretation could be that many developers simply do not utilize or care about good practice, even if they "know better". This would imply that the experts are correct in their recommendations of what constitutes "good practice", but in the real world the benefit to individual developers in using this best practice is limited. This would indicate a need for experts and researchers to study the return on investment of applying good practice, not only on a company level, or for a product, but also for the individual developer. Issues such as demands for productivity or time-to-market might be prioritized.

A third interpretation, and the worst possible one, could be that many of the respondents simply do not understand the concepts of object-orientation, and thus do not understand the necessity of following best practice put forward by experts. This would indicate a fundamental failure in terms of the training and education of developers, but also in the ability of experts and researchers to spread their knowledge, and the benefits of adhering to good practice, to industry and practice.

A puristic interpretation of the survey results at this time is most likely counterproductive and premature. Further in-depth analysis and study is needed, both of the results of the survey, but also by other researchers performing their own empirical studies. The central conclusion that can be drawn from the results in this survey, however, is that the assumptions of *what* constitutes good practice and *how* the object-orient concepts are interpreted and *used* may be flawed and must be studied in further detail.

## 6. CONCLUSIONS

We have presented some of the results of a large-scale empirical study of software developers. Our overall goal is to better understand how developers make design decisions, in particular how they understand and apply the expert advice, or "theory", on good object-oriented design. We carried out our study through an on-line survey, which was advertised through a variety of means including YouTube and Twitter. In this paper, we focused on the use of encapsulation, class size as measured by number of methods, and depth of classes in the inheritance hierarchy.

While our main contribution in this paper is the results of the survey, we also regard our methodology as an important contribution. The methodology consists of how to use and design survey concepts, as well as enable large scale participation. We are not aware of any other attempt to determine what advice software developers follow, certainly not on the scale of the study presented in this paper. Our results demonstrate that the methodology is sound, and we encourage other researchers to use it for other similar studies.

A total of 4823 respondents started the survey, and 3785 completed all the mandatory questions, a completion rate of 78.5%. The results indicate that developers generally, but not universally, follow the advice on the importance of hiding representation. Regarding the importance of observing, monitoring and limiting class depth and size, there is a large inconsistency among the developers. About half are for limitation, half are not. The half that are for limitation however agree with the experts' advice and generally choose similar limits to the experts.

While there is agreement to some degree with the advice regarding the studied concepts, our results make it quite clear that there is a significant proportion of software developers that do not follow the advice. Many may have suspected this, but our results provide very convincing evidence and not only confirms the suspicions, but also shows the level of the problem. The question to answer now is why — is it the theory, is it the training, or is it something else?

## Acknowledgements

We would like to thank everyone who participated in our survey, especially those who encouraged others to do so, and in particular Nat Torkington and Tim O'Reilly.

## 7. REFERENCES

[1] G. Baxter, M. Frean, J. Noble, M. Rickerby, H. Smith, M. Visser, H. Melton, and E. Tempero. Understanding the shape of Java software. In W. Cook, editor, *OOPSLA*, pages 397–412, Oct. 2006.

[2] E. V. Berard. *Essays on object-oriented software engineering (vol. 1)*. Prentice-Hall, Inc., 1993.

[3] G. Booch. *Object-Oriented Analysis and Design: with Applications*. Addison-Wesley, 2nd edition, 1994.

[4] F. Brito e Abreu and W. Melo. Evaluating the impact of object-oriented design on software quality. In *METRICS '96: Proceedings of the 3rd International Symposium on Software Metrics*, page 90, Washington, DC, USA, 1996. IEEE Computer Society.

[5] M. Bunge. *Treatise on Basic Philosophy: Ontology I: The Furniture of the World*. Springer, 1977.

[6] S. R. Chidamber and C. F. Kemerer. A metrics suite for object oriented design. *IEEE Trans. Softw. Eng.*, 20(6):476–493, 1994.

[7] K. El Emam, S. Benlarbi, N. Goel, W. Melo, H. Lounis, and S. N. Rai. The optimal class size for object-oriented software. *IEEE Trans. Softw. Eng.*, 28(5):494–509, 2002.

[8] N. E. Fenton and M. Neil. A critique of software defect prediction models. *IEEE Trans. Softw. Eng.*, 25(5):675–689, 1999.

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

[10] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns*. Addison Wesley Publishing Company, 1994.

[11] J. A. Gliner and G. A. Morgan. *Methods in Applied Settings: An Integrated Approach to Design and Analysis*. Lawrence Erlbaum Associates, 2000.

[12] T. Gorschek, P. Garre, S. Larsson, and C. Wohlin. A model for technology transfer in practice. *IEEE Softw.*, 23(6):88–95, 2006.

[13] L. Hatton. Reexamining the fault density-component size connection. *IEEE Softw.*, 14(2):89–97, 1997.

[14] A. Holub. Why getter and setter methods are evil: Make your code more maintainable by avoiding accessors. JavaWorld.com, Sept. 2003.

[15] R. E. Johnson and B. Foote. Designing reusable classes. *Journal of Object-Oriented Programming*, June/July 1988.

[16] M. Lorenz and J. Kidd. *Object-oriented software metrics: a practical guide*. Prentice-Hall, 1994.

[17] D. L. Parnas. On the criteria to be used in decomposing systems into modules. *Commun. ACM*, 15(12):1053–1058, 1972.

[18] T. Punter, M. Ciolkowski, B. Freimut, and I. John. Conducting on-line surveys in software engineering. In *ISESE '03: Proceedings of the 2003 International Symposium on Empirical Software Engineering*, page 80, Washington, DC, USA, 2003. IEEE Computer Society.

[19] A. Riel. *Object-oriented design heuristics*. Addison-Wesley, 1996.

[20] C. Robson. *Real World Research: A Resource for Social Scientists and Practitioner-Researchers*. Wiley-Blackwell, 2nd edition, 2002.

[21] N. Schärli, A. P. Black, and S. Ducasse. Object-oriented encapsulation for dynamically typed languages. In *Proceedings of the 19th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*, pages 130–149, New York, NY, USA, 2004. ACM.

[22] R. Sedgewick and K. Wayne. *Introduction to Programming in Java*. Addison-Wesley, 2006.

[23] C. Szyperski. *Component Software: Beyond Object-Oriented Programming*. Addison-Wesley, 2nd edition, 2002.

[24] E. Tempero. How fields are used in Java: An empirical study. In *Australian Software Engineering Conference (ASWEC)*, pages 91–100, Apr. 2009.

[25] E. Tempero, J. Noble, and H. Melton. How do Java programs use inheritance? an empirical study of inheritance in Java software. In J. Vitek, editor, *22nd European Conference on Object-Oriented Programming (ECOOP)*, pages 667–691, Paphos, Cyprus, July 2008. Springer Berlin / Heidelberg.

[26] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, R. B., and A. Wesslén. *Experimental Software Engineering – An Introduction*. Kluwer Academic Publishers, 2000.