# Search-based Refactoring: Metrics are Not Enough

Chris Simons[1], Jeremy Singer[2], and David R. White[2]

[1] Dept. of Computer Science and Creative Technologies, University of West England, Bristol BS16 1QY, UK
`chris.simons@uwe.ac.uk`

[2] School of Computing Science, University of Glasgow, G12 8RZ, UK.
`{jeremy.singer,david.r.white}@glasgow.ac.uk`

**Abstract.** Search-based Software Engineering (SBSE) techniques have been applied extensively to refactor software, often based on metrics that describe the object-oriented structure of an application. Recent work shows that in some cases applying popular SBSE tools to open-source software does not necessarily lead to an improved version of the software as assessed by some subjective criteria. Through a survey of professionals, we investigate the relationship between popular SBSE refactoring metrics and the subjective opinions of software engineers. We find little or no correlation between the two. Through qualitative analysis, we find that a simple static view of software is insufficient to assess software quality, and that software quality is dependent on factors that are not amenable to measurement via metrics. We recommend that future SBSE refactoring research should incorporate information about the dynamic behaviour of software, and conclude that a human-in-the-loop approach may be the only way to refactor software in a manner helpful to an engineer.

**Keywords:** Search-based Software Engineering, Metrics, Optimisation, Software Quality.

## 1 Motivation

Search-Based Software Engineering (SBSE) has been extensively applied to refactor object-oriented software based on metrics that quantify structural properties of an object-oriented design, such as measures of cohesion, coupling, the number of classes and the nature of the object hierarchy (e.g. [20], [22], [27], [31]).

Recent work shows that applying SBSE refactoring using metrics to the open-source Apache Ant project [8] does not result in an improved design as assessed by an expert. Other work reveals that when refactoring is conducted using a number of cohesion metrics, there can be disagreement between the metrics [32].

This raises the question of whether metrics are a good guide to software quality, at least in the context of refactoring. If metrics indicate that a refactored design is improved, but an engineer does not perceive any improvement, how is the engineer judging the quality differently? That is, metrics are essentially a proxy for something described as *software quality*, but what exactly is it that we are trying to find a proxy for? And if we can answer that question, is the correlation between the metrics we are using and the quality that we trying to optimise well-established?

In this paper, we try to answer these questions, by placing the ultimate judgement of software quality in the hands of expert industrial software engineers, by measuring the correlation between metrics used in SBSE and human judgement, and by examining the articulated justifications for those judgements.

## 2 Hypothesis

We formulate our hypotheses such that the null hypothesis makes no assumption of an effect:

- $H_0$: There is no correlation between software metric values and software engineer evaluation of quality for a given software design.
- $H_1$: There is a correlation between software metric values and software engineer evaluation of quality for a given software design.

## 3 Survey Design

The goal of our survey is to ask software engineers for their evaluation of a set of software designs in order to compare their impressions of design quality with scores derived from metrics. We chose to conduct a survey of software engineers by means of an online questionnaire. The selection of appropriate software designs, design qualities, metrics and engineers for the questionnaire requires careful consideration. Thus drawing upon best survey practice (e.g. [13], [14], [17], [29]), each of these components is described as follows.

### 3.1 Selection of Software Designs

We included two problem domains to strengthen the generality of our findings. Examples needed to be large enough to be meaningful to an engineer whilst not imposing excessive cognitive load and hence fatigue. We sought industrial benchmark designs, but none were readily available, and therefore two standard examples were selected: the Automated Teller Machine [12] and a nautical cruise booking system [5]. To reduce cognitive load, we used the Unified Modelling Language (UML) [33], widely heralded as “the lingua franca” of object-oriented modelling [35]. We explored the idea of using refactoring tools (e.g. CodeImp [31]) to generate designs, but no available tool met our requirements. Thus we invited five experienced software engineers to produce class designs for the two problems. Minor adjustments were made to the ten designs to ensure both a consistent level of abstraction and a range of metric scores.

### 3.2 Selection of Design Qualities

There exist a variety of software design quality definitions, including those from the International Standards Organisation and International Electrotechnical Commission (ISO/IEC 25010:2011) [21]. We drew inspiration from studies relating design qualities to corresponding properties and metrics (e.g. [24]). Bansiya and Davis [7] propose a Quality Model for Object-Oriented Design (QMOOD) that is popular in the literature and offers a hierarchical model of qualities, properties and metrics together with mappings between them. QMOOD defines six qualities: reusability, flexibility, understandability, functionality, extendibility and effectiveness (see [7], Table 1). While such quality attributes are good candidates for use in our survey, we considered that evaluating six qualities would place an unreasonable cognitive load on respondents, and so we focused on the most problem-domain independent qualities i.e. the first three. The Bansiya and Davis definitions of reusability, flexibility and understandability are given in Table 1.

### 3.3 Selection of Metrics

In order to select suitable metrics for the experiment, we conducted a simple review of the SBSE literature. We drew upon previous metrics reviews [24], [38] and guidelines for conducting systematic literature surveys [25], [26].

We addressed the question: *what is the distribution of software metrics among the SBSE refactoring literature?* We chose to rely upon the SBSE Repository [39] and through a simple search query of "software metrics", extracted 57 papers. We narrowed this list, by examining their titles, abstracts, introductions and conclusions. Examples of excluded sources included papers on software defect prediction and effort estimation. This process yielded 23 papers.

Our analysis reveals 118 different metrics used in the 23 sources. The metrics relate mostly to structural integrity such as cohesion and coupling, but measures of design size, complexity and elegance are also present. Of the 118 metrics, only three individual metrics — Lack of Cohesion of Methods (LCOM) and its variants, Module Quality (MQ) and Evaluation Metric (EVM) — are reported in

**Table 1.** Bansiya, Davis [7] Definitions of Reusability, Flexibility, Understandability

| Quality Attribute | Definition |
|---|---|
| Reusability | Reflects the presence of object-oriented design characteristics that allow a design to be reapplied to a new problem without significant effort. |
| Flexibility | Characteristics that allow the incorporation of changes in a design. The ability of a design to be adapted to provide functionally related capabilities. |
| Understandability | The properties of a design that enable it to be easily learned and comprehended. This directly relates to the complexity of the design structure. |

more than one paper; the use of only one suite of metrics (QMOOD) is reported twice. These results are illustrated in Table 2.

In selecting the metrics to be evaluated, we considered that those found in multiple were a reflection of wider use in the field. We first considered LCOM, MQ and EVM. However, LCOM relies on knowledge of implementation dependencies e.g. what methods use which attributes in a class, while MQ and EVM are metrics for dependency-based module clustering. All typically require programming language source code for calculation. Alternatively, the QMOOD metrics suite is used in two studies, and the metrics may be mapped to design quality attributes [7], which is consistent with our chosen hypothesis.

Inspection of the 11 QMOOD metrics reveals that their straightforward computation makes them amenable to evaluation on both human-produced design models and programming language source code. Closer analysis reveals that 3 of the metrics i.e. Design Size in Classes (DSC), Direct Class Coupling (DCC) and Number of Methods (NOM) are readily applicable to the human-generated design models. Metrics relating to inheritance (Number Of Hierarchies, NOH, Average Number of Ancestors, ANA) are excluded as only six of the final designs show inheritance, with only one class hierarchy each.

Thus DSC, DCC and NOM have been selected from the QMOOD suite for use in the experiment. Two other metrics were selected: Numbers among classes (NAC) [37], and Numbers of Attributes and Methods (NOAM). NAC was chosen as it had been used previously by Barros and Farzat [8] to relate design improvements to understandability. The NOAM metric was introduced to cater for any influence of the number of attributes in a design, which was otherwise lacking. Both NAC and NOAM are readily calculated for human-generated designs.

## 3.4 Correlation Analysis

We use correlation analysis to compare the chosen metrics to the opinions of engineers. Whether our conclusions apply to the derivation of high-level properties from metrics in suites such as QMOOD depends greatly on how those suites use metric values. For example, in a weighted-sum method there may be a trade-off between two metrics in defining a high-level quality, and thus correlation analysis may be insufficient to validate such an approach. However, the weightings involved may be somewhat arbitrary — how can you compare an increase in

**Table 2.** Metrics and Papers Reporting Use

| Metrics | Papers Reporting Use |
|---|---|
| MQ | Azar, 2009 [6], Glavas, 2011 [15], Glavas, 2011b [16], Harman, 2004 [18], Harman, 2005 [19] |
| EVM | Barros, 2013 [8], Harman, 2005 [19] |
| LCOM and variants | Azar, 2009 [6], Barros, 2013 [8], Ó Cinnéide, 2012 [32], Koc, 2012 [28] |
| QMOOD Suite | Jensen, 2010 [22], O'Keeffe, 2007 [34] |

design size against a reduction in cohesion? The range of metric values is also problem-specific, therefore it is likely that one metric will dominate another, and a simple correlation analysis will be sufficient in such cases. To fully evaluate QMOOD and similar approaches, it would be necessary to evaluate all of the metrics employed in any given suite, which is impractical in a survey of this type, and thus we must focus on the evaluation of individual metrics.

### 3.5 Target Population

In selecting the target population of software engineers, we concluded that industrial practitioners would provide the best audience given the aim of the survey. We considered that academics should not be targeted as their design experience might be confined to educational software design. The sampling frame of the target population was achieved by inviting members of the Association of C and C++ Users (ACCU) [1] and the British Computer Society (BCS) [10] — two organisations prominent in promoting professionalism and best practice in software design and development. We investigated the possibility of using Amazon Mechanical Turk (AMT) [4], but this required US residency.

## 4 Questionnaire Design

We wished to present a design of meaningful size and complexity without cognitively overloading the engineer. We chose two case studies and five solutions to each; the designs are online [36], together with their calculated metrics values (DSC, DCC, NOM, NAC and NOAM). Designs were presented at random to the participants, one model per participant. To capture their judgement of design qualities, we used a Likert Scale with seven levels: "strongly disagree", "disagree", "somewhat disagree", "neutral", "somewhat agree", "agree" and "strongly agree". We asked participants to provide rationale for their judgements, and analysed responses using thematic coding. We requested a participant's number of years of design and development experience, as well as their opinion of their own design expertise and confidence on the same Likert Scale.

Informed consent was obtained from participants prior to conducting the questionnaire and participant withdrawal was possible at any point. Any personal data recorded, e.g. email addresses, was entirely at the discretion of the participant. All survey information was strictly confidential and published survey information is reported either as aggregate data or is anonymised.

Full details of our questionnaire design are available [36]. Pretesting (see [13], Chapter 10) was conducted. Five experienced software engineers undertook the draft questionnaire and their response experiences evaluated with respect to:

- *Assimilation:* respondents reported being able to satisfactorily attend to and remember relevant questions and instructions.
- *Comprehension:* respondents also reported being able to understand the questions and definitions of the three design qualities.

– *Recall:* some respondents found difficulty in recalling a design when completing their evaluation. As a result, “back” buttons were made more explicit.
– *Reporting:* respondents reported that Likert scales and the free text response answers were clear and offered adequate opportunity for responses.

We used SurveyGizmo as our survey platform, as it supports sufficient logic to enable the randomisation of problem description and model assignments we required. As an incentive for participation, we also offered participants the opportunity to be entered into a prize draw.

## 5 The Survey Process

Our survey was open from 18 January to 28 February 2015, a total of 42 days. Invitations to participate were dispatched to approximately 900 ACCU [1] members and 200 members of the Bristol (UK) branch of the BCS [11]. An invitation to participate in the survey was also posted on a discussion forum via the BCS Members' Group on LinkedIn [30] to which approximately 11,000 BCS practitioners have access, although the number of regular contributors is much lower.

Some lively comments were posted to the LinkedIn forum. Despite piloting the questionnaire, a number of participants posted feedback stating that it was not realistic to form an impression of design qualities using a UML class model in isolation from other aspects of software development e.g. dynamic models of behaviour, requirements, test plan etc. (although this is exactly what search-based refactoring is doing). One forum contributor remarked to the authors of the survey: *“it seems your idea of what quality is and how to judge it is not the same as many of us in the industry,”* foreshadowing our survey results.

## 6 Results

A total of 50 responses was received, although we discarded one of these (see Section 8). We were pleased to receive 50 responses and we are grateful to those engineers who gave up the time to carefully examine the case studies.

The distribution of engineers' experience is given in Figure 1, which illustrates that our respondents were on the whole very experienced. We asked them to rate their expertise in the area of software design, and their confidence in their own judgement — the results are shown in Figure 2. We are very confident of the experience and expertise of our respondents.

We divide our results between responses to the quantitative questions, where we asked respondents to rate the qualities of software on a Likert scale, and the qualitative answers, where asked them to justify their quantitative responses.

### 6.1 Quantitative Results

In order to test our hypothesis, we perform correlation analysis on the relationship between the responses of those surveyed and the selected metrics. Engineers

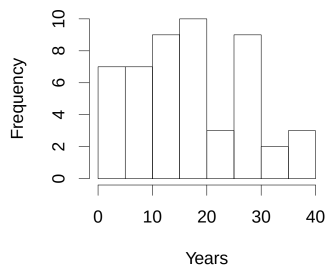

**Fig. 1.** Histogram of the Programming Experience of our Respondents

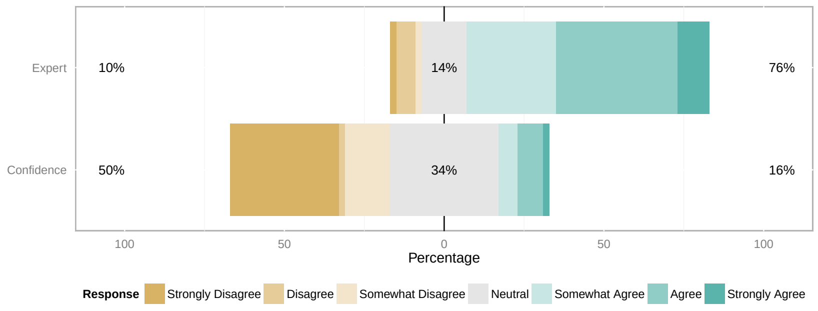

**Fig. 2.** Respondents' Self-Assessed Significant Expertise in Software Design and Confidence in their Ratings of the Models

were asked three questions: whether they agreed that the design was Understandable, Reusable, and Flexible. These questions required a Likert response on a seven-step scale from "strongly disagree" to "strongly agree". We plotted responses against the corresponding value of each software metric for that design.

Figure 3 gives correlation plots for each quality. A score of one corresponds to "strongly disagree". Table 3 gives the Spearman's Rank Coefficient for each correlation. We observe from these results that there is *almost no correlation between the perception of software engineers and the metrics.* We then performed a two-sided significance test against the Spearman's Rank coefficients, and the p-values are given in Table 4. No correlations were significant at the 0.05 level.

Thus, we are unable to refute our null hypothesis $H_0$. This is quite a significant result: if we assume that our experimental results are valid (see Section 8 for threats to validity) and that they generalise to other object designs, then the metrics we are examining are *not helpful in improving these software qualities.*

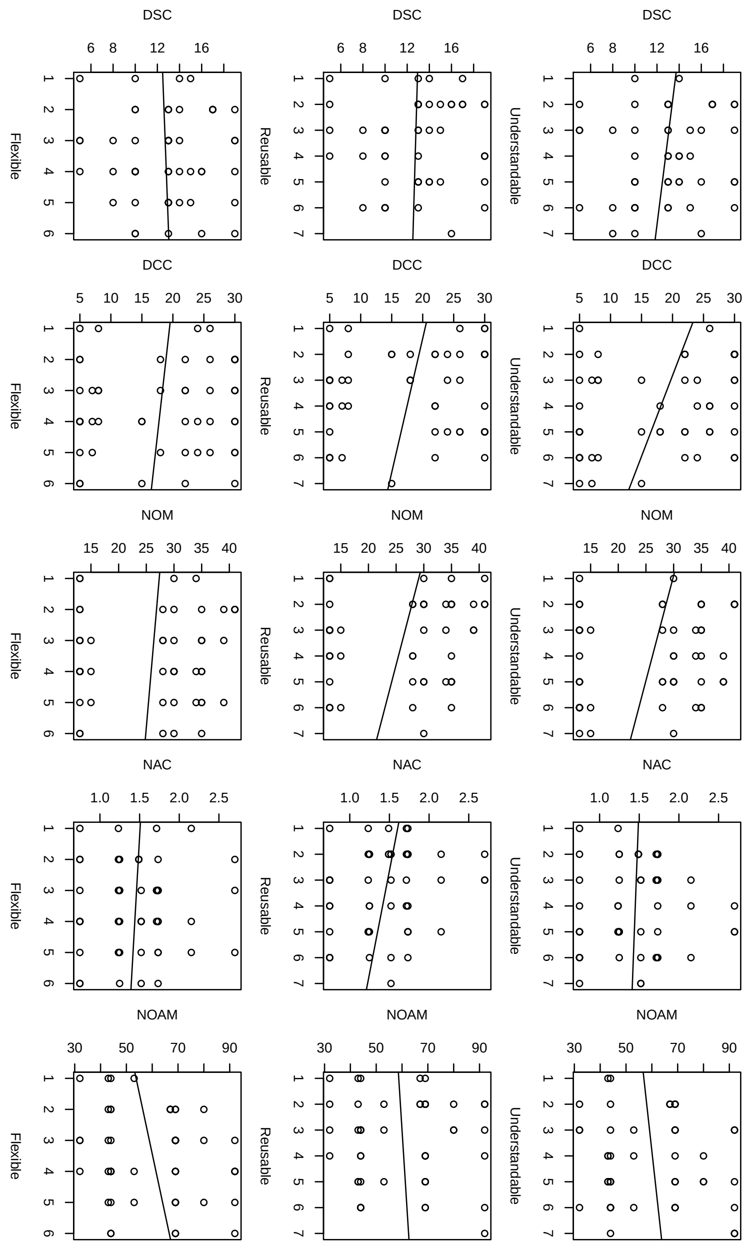

**Fig. 3.** Correlation between Expert Opinion of Three Software Qualities and Five Software Metrics

| Quality | DSC | DCC | NOM | NAC | NOAM |
|---|---|---|---|---|---|
| Understandable | -0.128 | -0.271 | -0.203 | -0.0400 | 0.103 |
| Reusable | -0.0280 | -0.158 | -0.195 | -0.200 | 0.0572 |
| Flexible | 0.0386 | -0.0806 | -0.0677 | -0.0613 | 0.202 |

**Table 3.** Spearman's Rank Coefficients for the Correlation between Our Metrics and Human Judgement (to 3 s.f.)

| Quality | DSC | DCC | NOM | NAC | NOAM |
|---|---|---|---|---|---|
| Understandable | 0.375 | 0.0571 | 0.156 | 0.783 | 0.478 |
| Reusable | 0.847 | 0.272 | 0.174 | 0.163 | 0.693 |
| Flexible | 0.790 | 0.578 | 0.640 | 0.672 | 0.160 |

**Table 4.** P-Values for a Two-Sided Test of the Spearman's Rank Correlation Coefficients in Table 3 (to 3 s.f.)

### 6.2 Qualitative Results

We were intrigued as to why there was no correlation between the quality metrics and the judgement of software engineers. In particular, we wished to understand the thought processes of software engineers: what makes one design better than another? We asked the engineers to justify their judgements in prose.

We then coded their responses as per *grounded theory* ([14] Ch. 4, [29] Ch. 11), an inductive process where text responses are categorised in order to derive trends from textual responses. This process is somewhat subjective; two of the authors coded the data independently and then resolved their differences through comparison and discussion. We divide the coding results between the three questions, and the results are given in Tables 5, 6 and 7. The frequency column indicates the number of responses that fell into a given category; a single response may fall into multiple categories. We make the following observations:

| Coding | Frequency |
|---|---|
| Needs something more (dynamics, context, reqts, rationale etc.) | 22 |
| Incorrect or unclear responsibility assignment | 13 |
| Clear traceability to problem domain | 10 |
| Clear breakdown of purpose | 6 |
| Clear element naming | 5 |
| Missing abstractions | 4 |
| No response or no explanation | 3 |
| Poor layout | 1 |

**Table 5.** Classifications for Rationale behind Judgement of "Understandable"

| Coding | Frequency |
|---|---|
| Parts of the design should be easy to modify | 12 |
| Problem specific | 11 |
| Needs something more (dynamics, context, reqts etc.) | 8 |
| Incorrect or missing abstractions | 8 |
| Class coupling | 5 |
| Incorrect / unclear responsibility assignment | 2 |
| Separation of concerns | 2 |
| Hard to test | 1 |
| Simplistic | 1 |
| No response / no clear explanation | 8 |

**Table 6.** Classifications for Rationale behind Judgement of "Flexible"

| Coding | Frequency |
|---|---|
| Problem specific | 24 |
| Parts of the design are reusable, others not | 18 |
| Class coupling | 5 |
| Needs something more (dynamics, context, reqts etc.) | 5 |
| Incorrect abstractions | 3 |
| Lack of object-oriented design | 2 |
| Separation of concerns | 2 |
| OO languages | 1 |
| Simplistic | 1 |
| No response | 1 |

**Table 7.** Classifications for Rationale behind Judgement of "Reusable"

**A Class Diagram is Not Enough** The responses reflect the conversation on the professional network LinkedIn (see Section 5). It is clear that our respondents are unable to assess the design of a system based solely on the class diagram; for example, one respondent commented on their ability to understand the system:

> "[A class diagram] offers no information about the interactions, procedures or validations and therefore is a really superficial view of the software design"

We agree with this comment, and we are unaware of any object-oriented SBSE refactoring that takes as input a description of the dynamic behaviour of a system design. How can we expect to imitate the judgement of a software engineer if we do not take into account the same information? The most commonly request was for information about system behaviour.

**The Problem Domain Matters** Another trend was the relationship between the design and the problem domain: from fundamental decisions such as the names given to objects and associations, to the high-level mapping between requirements and the model, it is clear that judging the model in isolation from the problem domain is not meaningful to an engineer. Many comments referred to the problem domain, or the way in which the problem had been analysed.

**Qualities have Meaning only in a given Context** When considering a given quality, there is a need for context. Many comments were along the lines of "Reusable for what purpose?" or "Reusability depends on context…". It is not enough to look at the high-level structure alone:

> "Reusability is about more than just a class diagram and function definition. It's about how the software is actually written and this can be good or bad."

> "Re-use only has meaning if it itself is defined/scoped, tested and managed as part of the project."

We may also consider parts of a model as representing distinct contexts: respondents pointed out that the qualities may vary across different parts of the model. It is perhaps naive to expect to holistically optimise a system.

**Good Design is Intuitive** There was a clear difficulty amongst some respondents in articulating their thought processes, as if it involved some aesthetic judgement: for example, some replies included: "My guts says no", "Seems to fit together well", "intuitive". Such responses appear to resonate with the design patterns notion of *Quality Without a Name* [2], generally taken to mean that good quality is something that can be recognised, but difficult to describe in words: i.e. "you know it when you see it".

**Our Standard Metrics play a (Minor) Part** The aspects of the model that could be judged by our respondents do indeed overlap with the intentions of our chosen metrics. For example, engineers consider the object hierarchy, separation of concerns, cohesion, and division of responsibilities as important elements in judging design quality. However, these properties are given minor consideration in their responses compared to the central concerns mentioned above.

## 7 Related Work

Although examples of related empirical investigations are not abundant in the literature, Katmarksi and Koschke [23] conducted an empirical study to investigate whether complexity metrics agree with programmer opinion. They selected examples of Java source code for programmer complexity evaluation, and applied a variety of control-flow and data-flow complexity metrics. Among 200 participants, they report some matching of programmer ranking with complexity metrics, although participants were presented with small Java methods (12–51 lines of code) and object-oriented design aspects were ignored. Moreover, the 200 participants were drawn from a sample of students and academics, and Katmarksi and Koschke suggest that *"professional programmers might report differently"*.

Ó Cinnéide et al. [32] conducted a rigorous investigation into refactoring metrics — rather than refactoring source code, their goal was to assess the software metrics that guide the automated refactoring through repeated refactoring experiments. They investigated five cohesion metrics using eight real-world Java systems, involving 300,000 lines of code and over 3,000 refactorings. The authors report that the cohesion metrics disagreed with each other in 55% of cases. It is also interesting to note the disagreement among metrics alongside the disagreement in software engineers' impressions of software qualities in this experiment.

Recent studies by Bavota et al. [9] investigate possible relationships between metrics, code smells and developer refactoring activities. They mined the evolution history of three Java open source projects to investigate whether refactoring activities occur on code components for which certain indicators such as quality metrics or the presence of smells as detected by tools suggest there might be need for refactoring operations. Their results indicate that, more often than not, quality metrics do not show a clear relationship with refactoring. In other words, refactoring operations are generally focused on code components for which quality metrics do not suggest there might be need for refactoring operations. Such findings do seem to resonate with the lack of correlation between quality metrics and software engineer impressions of software qualities found in this experiment.

## 8 Threats to Validity

The biggest weakness of the survey is that the class models are small educational examples. This was constrained by two factors: the screen space used to display designs, and the cognitive overhead on respondents to understand the requirements and design sufficiently to answer our questions. As a consequence, we excluded metrics that are only meaningful when considering larger models (for example, those measuring the inheritance hierarchy). We spent much time selecting and refining our case studies, after considering various approaches and problems, and decided upon those that we considered to be as large as the aforementioned restrictions would allow.

The qualities are often discussed informally within software engineering without attention to their precise meaning. To reduce the risk of ambiguity, we included definitions within the survey. Respondents were free to move between the survey pages, meaning they could return to the definitions as necessary, and indeed some respondents quoted our definitions within their rationale.

With any survey, there is a danger of bias in terms of the respondent population. We tried to avoid this by targeting a specific set of organisations directly relevant to the study, i.e. software engineers, through a set of professional institutions. One final lesson we learnt from the process is that we did not pilot the survey sufficiently. For example, it transpired that the survey did not work on all mobile devices, which may have skewed our sample.

## 9 Conclusions

### 9.1 Refactoring Metrics are not correlated with Human Judgement

In Section 6.1, we were unable to refute the null hypothesis that there is no correlation between these standard SBSE refactoring metrics and software quality. Thus we are unable to support the conjecture that SBSE refactoring tools relying solely on these metrics will consistently propose useful refactored models to engineers. Whilst we dealt with only a subset of available metrics and two example problems, our qualitative results suggest that this statement will generalise to other metrics and problem domains, simply because metrics do not take into account the information required to make a sound judgement.

### 9.2 Wider Lessons Regarding Refactoring

We conclude from our qualitative analysis that metrics based on an object-oriented design are insufficient to optimise software quality. Furthermore, no simple metrics will be able to entirely capture the essential aspects of a software design used by human engineers when making judgements of software qualities.

Software is inextricably connected to a problem domain, and attempting to treat the design of a system in isolation disregards this connection. We note the recent advances in Machine Learning and automatic programming to address such concerns e.g. [3], and do not discount that such developments could one day solve the problem of machine judgement of software quality. However, without such advances we recommend that, although not novel, *human-in-the-loop* systems are the only viable method for automated refactoring tools that produce meaningful solutions.

Future work could include repeating the exercise but including some form of design concerning the dynamic behaviour of each system. The lack of a dynamic view (for example, a statechart) was the most common complaint amongst respondents, and it would be interesting to see how this affects their ability to discern the qualities of the software design, as well as the impact it may have on the articulation of their rationale. However, we suspect that such an experiment may be subject to very low response rates, and we recommend a smaller focus-group approach involving observed exercises and interviews.

Omitting information that could deanonymise a respondent, the data gathered and used to produce the figures in this paper are online [36].

## 10 Acknowledgements

We would like to thank our colleagues at UWE for their help in the design of the survey; Mel Ó Cinnéide and Iman Hemati Moghadam for generously sharing code; Per Runeson for his insightful advice; the professional organisations the ACCU and BCS for permitting mailshots to their members; and we are very especially grateful to all those who took the time to respond to the survey.

## References

1. ACCU: Association of C and C++ Users. http://www.accu.org/, Accessed: 3rd June 2015
2. Alexander, C.: The Timeless Way of Building. Vol. 1. Oxford University Press (1979)
3. Amal, B., Kessentini, M., Bechikh, S., Dea, J., Ben Said, L.: On the Use of Machine Learning and Search-Based Software Engineering for Ill-Defined Fitness Function: A Case Study on Software Refactoring. In: SSBSE 2014 (2014)
4. AMT: Amazon Mechanical Turk. http://www.mturk.com/mturk/welcome/, Accessed: 3rd June 2015
5. Apperly, H., Hofman, R., Latchem, S., Maybank, B., McGibbon, B., Piper, D., Simons, C.: Service- and Component-based Development. Addison-Wesley (2003)
6. Azar, D., Harmanani, H., Korkmaz, R.: A Hybrid Heuristic Approach to Optimize Rule-based Software Quality Estimation Models. Information and Software Technology 51(9), 1365–1376 (2009)
7. Bansiya, J. and Davis, C.G.: A Hierarchical Model for Object-Oriented Design Quality Assessment. IEEE Trans. on Software Engineering 28(1), 4–17 (2002)
8. Barros, M., Farzat, F.: What Can a Big Program Teach Us about Optimization? In: SSBSE 2013, LCNS 8084. Springer (2013)
9. Bavota, G., De Lucia, A., Di Penta, M., Oliveto, R., Palomba, F.: An Experimental Investigation on the Innate Relationship between Quality and Refactoring. Journal of Systems and Software 107, 1–14 (2015)
10. BCS: British Computer Society. http://www.bcs.org/, Accessed: 3rd June 2015
11. BCS-BRISTOL: British Computer Society Bristol Branch. http://www.bristol.bcs.org.uk/, Accessed: 3rd June 2015
12. Bjork, R.: ATM Simulation. http://www.math-cs.gordon.edu/courses/cs211/ATMExample/, Accessed: 3rd June 2015
13. Blair, J., Czaja, R., Blair, E.: Designing Surveys: A Guide to Decisions and Procedures. Sage Publications (2014)
14. Gibbs, G.: Analysing Qualitative Data. Sage Publications (2007)
15. Glavas, G., Fertalj, K.: Metaheuristic Approach to Class Responsibility Assignment Problem. In: Proceedings of the ITI 33rd International Conference on Information Technology Interfaces (ITI). IEEE (2011)
16. Glavaš, G., Fertalj, K.: Solving the Class Responsibility Assignment Problem using Metaheuristic Approach. Journal of Computing and Information Technology 19(4), 275–283 (2011)
17. Groves, R., Fowler, F., Couper, M., Lepkowski, J., Singer, E., Tourangeau, R.: Survey Methodology. Wiley (2004)
18. Harman, M., Clark, J.: Metrics are Fitness Functions too. In: Proceedings of the 10th International Symposium on Software Metrics. IEEE (2004)
19. Harman, M., Swift, S., Mahdavi, K.: An Empirical Study of the Robustness of two Module Clustering Fitness Functions. In: GECCO 2005. ACM (2005)
20. Harman, M., Tratt, L.: Pareto Optimal Search Based Refactoring at the Design Level. In: GECCO 2007. ACM (2007)
21. ISO/IEC: Standard 25010:2011. http://www.iso.org/iso/catalogue_detail.htm?csnumber=35733, Accessed: 3rd June 2015
22. Jensen, A., Cheng, B.: On the use of Genetic Programming for Automated Refactoring and the Introduction of Design Patterns. In: GECCO 2010. ACM (2010)

23. Katzmarski, B., Koschke, R.: Program Complexity Metrics and Programmer Opinions. In: Proceedings of the 20th International Conference on Program Comprehension (ICPC). IEEE (2012)
24. Khan, Y., Khararah, O.: A Systematic Review on the Relationships between MOOD/QMOOD Metrics and External Software Quality Attributes. Tech. rep., Department of Information and Computer Science, King Fahd University of Petroleum and Minerals, Dhahran, Saudi Arabia (2014)
25. Kitchenham, B.: Guidelines for Performing Systematic Literature Reviews in Software Engineering. Tech. Rep. EBSE-2007-01, School of Computer Science and Mathematics, Keele University, Keele, Staffs, ST5 5BG, United Kingdom (2007)
26. Kitchenham, B., Mendes, E., Travassos, G.: A Systematic Review of Cross-vs. Within-company Cost Estimation Studies. In: Proceedings of the 10th International Conference on Evaluation and Assessment in Software Engineering. British Computer Society (2006)
27. Koc, E., Ersoy, N., Andac, A., Camlidere, Z., Cereci, I., Kilic, H.: An Empirical Study about Search-Based Refactoring using Alternative Multiple and Population-Based Search Techniques. In: Computer and Information Sciences II, pp. 59–66. Springer (2012)
28. Koc, E., Ersoy, N., Camlidere, Z., Kilic, H.: A Web-service for Automated Software Refactoring using Artificial Bee Colony Optimization. In: Advances in Swarm Intelligence, pp. 318–325. Springer (2012)
29. Lazar, J., Feng, J., Hochheiser, H.: Research Methods in Human-Computing Interaction. Wiley (2010)
30. LinkedIn: LinkedIn Professional Network. `https://uk.linkedin.com/`, Accessed: 3rd June 2015
31. Moghadam, I., Ó Cinnéide, M.: Code-Imp: A Tool for Automated Search-Based Refactoring. In: Proceedings of the 4th Workshop on Refactoring Tools (WRT '11). ACM Press (2011)
32. Ó Cinnéide, M., Tratt, L., Harman, M., Counsell, S., Moghadam, I.: Experimental assessment of software metrics using automated refactoring. In: Proceedings of the ACM-IEEE international symposium on Empirical software engineering and measurement. ACM (2012)
33. ObjectManagementGroup: Unified Modelling Language. `http://www.uml.org/`, Accessed: 3rd June 2015
34. O'Keeffe, M., Ó Cinnéide, M.: Automated Design Improvement by Example. Frontiers in Artificial Intelligence and Applications 161, 315 (2007)
35. Petre, M.: UML In Practice. In: Proceedings of the International Conference on Software Engineering (ICSE). IEEE (2013)
36. Simons, C., Singer, J., White, D.R.: Survey Materials and Data. `http://www.cems.uwe.ac.uk/~clsimons/MetricsAreNotEnough/`, Accessed: 3rd June 2015
37. Simons, C., Parmee, I.: Elegant Object-Oriented Software Design via Interactive, Evolutionary Computation. IEEE Trans. on Systems, Man, and Cybernetics, Part C: Applications and Reviews 42(6), 1797–1805 (2012)
38. Sjoberg, D., Anda, B., Mockus, A.: Questioning Software Maintenance Metrics: a Comparative Case Study. In: Proceedings of the ACM-IEEE International Symposium on Empirical Software Engineering and Measurement. ACM Press (2012)
39. Zhang, Y.: SBSE Repository. `http://crestweb.cs.ucl.ac.uk/resources/sbse_repository/`, Accessed: 3rd June 2015