Mika V. Mäntylä. 2005. An experiment on subjective evolvability evaluation of object-oriented software: Explaining factors and interrater agreement. In: Proceedings of the 4th International Symposium on Empirical Software Engineering (ISESE 2005). Noosa Heads, Queensland, Australia. 17-18 November 2005, 10 pages.

An Experiment on Subjective Evolvability Evaluation of Object-Oriented Software: Explaining Factors and Interrater Agreement

Mika V. Mäntylä
Helsinki University of Technology, Software Business and Engineering Institute
P.O. Box 9210, FIN-02015 HUT, Finland
mika.mantyla@hut.fi

Abstract

Recent trends in software development have emphasized the importance of refactoring in preserving software evolvability. We performed two experiments on software evolvability evaluation, i.e. evaluating the existence of certain code problems called code smells and the refactoring decision. We studied the agreement of the evaluators. Interrater agreement was high for simple code smells and low for the refactoring decision. Furthermore, we analyzed evaluators' demographics and source code metrics as factors explaining the evaluations. The code metrics explained over 70% of the variation regarding the simple code smell evaluations, but only about 30% of the refactoring decision. Surprisingly, the demographics were not useful predictors neither for evaluating code smells nor the refactoring decision. The low agreement for the refactoring decisions may indicate difficulty in building tool support simulating real-life subjective refactoring decisions. However, code metrics tools should be effective in highlighting straightforward problems, e.g. simple code smells.

1. Introduction

Software evolvability – the ease of further developing software – is an important quality attribute greatly dictating the future potential of any software system. In the past there was a strong emphasis on up-front-design for ensuring software evolvability. However, recent trends such as agile software development and extreme programming have highlighted refactoring – modifying the internal structure of software without affecting its observable behaviour – as a key factor for ensuring software evolvability. For example, Microsoft has recognized the constant need to modify existing software structure to ease future development. Therefore, Microsoft's Office division determined that 20% of development effort should be budgeted to code modification (pp. 280-281[3]).

An important issue concerning software evolvability is the decision when to perform refactoring. It seems likely that wrong refactoring decisions can do more harm than good. Fowler and Beck have come up with a term called *code smell* [4] to help software developers in recognizing problematic code. These code smells are general descriptions of bad code that are supposed to help software developers decide when the code needs refactoring. Fowler and Beck [4] claim that exact criteria for refactoring decisions cannot be given: *"no set of metrics rivals informed human intuition"*.

Thus, humans play an important role in making software refactoring decisions. Still, most of the work around refactoring has focused on tools and metrics, see [11] for details. There are a limited number of empirical studies and controlled experiments studying subjective software evolvability evaluation, i.e. refactoring decisions and the evaluation of the existence of code smells. We studied this topic at the source code method level. Two experiments were made with a different set of students in each. The participants evaluated the existence of certain code smells for each method and then stated whether the method should be refactored or not. Our first objective was to assess the interrater agreement, i.e. the extent to which evaluators agree. High interrater agreement is a positive indication of the reliability of the subjective evaluations. Lack of interrater agreement can mean that some evaluators are mistaken in their evaluations. The second objective was to study how factors, such as the evaluated code itself and the background of the evaluators, affect the evaluations. An analysis of these factors can help us find predictors for the code smell evaluations and the refactoring decisions, which can be used, e.g. in building tool support.

Section 2 summarizes the prior work on subjective evaluation of software evolvability. Section 3 presents the methodology. Section 4 introduces the results, and Section 5 presents the discussion. Finally, Section 6 provides the conclusions and direction for future work.

2. Related work

Shneiderman et al. (pp. 134-138 [15]) reported results

from using peer reviews in software code quality evaluation. They conducted three peer-review sessions each having five professional programmers with similar background and experience. Each programmer provided one of their best programs, which was then evaluated by the four other participants. The review was performed by answering 13 questions on a seven point ordinal scale. The questions varied from blank line usage and the chosen algorithm to the ease of further development of the program. The results showed that in half of the evaluations three out of four programmers agreed on the subjective evaluations (answers differed by one at most). However, in 43,1% of the evaluations the range was two or less. The researchers tried to explain this by speculating that the subjects misunderstood the questions or the scale. However, the study does not account for factors, such as differences in developers' opinions about the program design, structure, and style that also might explain the results.

Kafura and Reddy [7] studied the relationship between software complexity metrics and software maintainability. Maintainability was measured using subjective evaluations by system experts. However, no details are given on how these evaluations were collected from the individuals, and no data is provided of the evaluations. Therefore, it is difficult to assess the study any further. Regardless, the researchers conclude that the expert evaluations on maintainability were in conformance with the source code metrics.

Shepperd [14] validated the usefulness of information flow metrics on software maintainability by collecting the subjective opinions of the maintainers for 89 modules of aerospace software that totalled around 30 000 lines of code. Each developer of the maintenance team was individually asked to classify each module from one to four on an ordinal scale on the perceived difficulty of a hypothetical maintenance task. For 73% of the modules the range was one or less, and thus the researchers concluded that there was a strong correspondence between the individual evaluations. However, as no detailed data is given, it is difficult to assess the study in more detail.

Oman et al. [13, 16] report on the construction of a maintainability index. In this work the researchers used source code metrics to create polynomial regression models that measured software maintainability. They calibrated the maintainability models according to how well they correlated with the subjective evaluations of software maintainers. To do this the researchers acquired source code and maintainers' opinions on eight industrial software systems, ranging from 1000 to 10 000 lines of code [13]. After calibration, they performed a validation study where they again acquired opinions and source code on six industrial systems. In the validation study they also saw discrepancies where one engineer was more lenient and other more critical to the systems they were evaluating. Although the study [13] does not directly report this, it seems that there was only the opinion of a single individual per software system that was used in the initial creation of the metric and the validation performed. Therefore, it is difficult to effectively study the differences in human maintainability evaluations. However, the researchers continued in tuning their maintainability measure and performing tests on several industrial systems. Finally, the researchers concluded that the automatic assessment corresponds well to the perceived view of the experts [16].

Kataoka et al. [8] studied the usefulness of improving the software quality with refactoring and report on a comparison between human evaluation and software metrics. According to the researchers, the subjective evaluation of an expert on the effectiveness of refactorings correlated quite well with the improvement in coupling metrics. The drawback in the study is that the data set consists only of five refactoring cases and that only one developer evaluated the effectiveness of the refactorings.

Genero et al. [6] studied the maintainability of UML-class diagrams. The researchers show that subjective evaluation of understandability, analyzability, and modifiability of UML-diagrams correlated with various class level metrics. In a follow-up study [5] they show correlation between subjective complexity evaluation and both the time required to understand the UML-diagram (0,242), and objective code metrics based diagram classification (0,539). However, these studies are made with UML-diagrams and they also lack results on the interrater agreement of the evaluations.

Mäntylä et al. [10] studied code smell evaluations of twelve industrial developers. They discovered the developers make conflicting evaluations of the source code. They also discovered inconsistencies when comparing the evaluations to source code metrics. The weaknesses of the study are that the developers' evaluations were based on recollection of the modules they had primarily worked with, and the large size of the evaluated modules (between 15 and 65 thousand lines of code).

Five out of the seven referred studies are not made with object-oriented code, and there are drawbacks in the two studies made with object-oriented software. The drawback in [8] is that the data set consists only of five refactorings and that only one developer evaluated the effectiveness of the refactorings. In [10] there were limitations with using module level evaluations and basing the evaluations on recollections of the modules the developers had primarily worked with. Thus, there

is ample research space to be filled.

3. Methodology

Section 3.1 presents the research problem. Section 3.2 presents the experimental setting. Section 3.3 introduces the methods used to analyze the data.

3.1. Research problem and questions

The research problem of this study is: *Do human evaluations of software evolvability differ from one another, and what are the explaining factors behind the evolvability evaluations.* First, the research problem inquires whether there are differences between human evolvability evaluations. Second, it seeks to explain the factors behind the evolvability evaluations. Thus, the research problem was further divided to two research questions.

The first research question studies the interrater agreement of the evolvability evaluations. **Research question 1:** *Is there an interrater agreement in subjective evolvability evaluation?* When studying interrater agreement we must first consider whether there should be interrater agreement between evaluators. When asking people for the best way to spend their holiday, we might not expect high interrater agreement as people are likely to favour different holiday plans. However, we might expect high interrater agreement when we look at the judges of figure skating contests. Thus, the purpose of interrater agreement analysis is to study the amount of agreement between evaluators' evolvability evaluations. The *hypothesis* is high interrater agreement for all the three code smells evaluated and for the refactoring decision.

The second research question studies possible factors explaining the evolvability evaluations. In this study we have focused on factors from two primary sources: code metrics and evaluators' demographics. **Research question 2:** *How much of the evolvability evaluation of a software element can be explained by the measurable characteristics of the software element and the evaluator demographics?* The *hypothesis* is that characteristics of the evaluated element and the evaluators' demographics can explain most of the variation in the evaluations. The software element characteristics are expected to have greater impact than the demographic data.

3.2. Experimental setting

Two experiments with some differences were made. We refer to them as Experiments A and B. Experiment A was made first and Experiment B was performed a year later. Both experiments had identical software and documentation on which the evaluators based their evaluations.

3.2.1. Software under study. A small Java application with nine classes and 1000 NLOC of code was created for these experiments. To ensure fluctuation in the evaluations, some pieces of the software were programmed poorly on purpose. The application was a family tree modelling software operating on relationships like spouse, parent, and child. Family tree modelling was chosen because the domain knowledge required in understanding the application is pretty simple. Ten methods were selected as the software elements for the evaluation.

3.2.2. Viewpoints of subjective evaluations

Experiment A. Four questions of each method were presented to get different viewpoints of the subjective evaluations. Three of the four questions focused on the existence of the following code smells [4]: Long Method, Long Parameter List, and Feature Envy, which were chosen because they can be studied at the method level. The fourth question asked if the method should be refactored to remove the smells.

Long method means that a method is too long and tries to perform many possibly unrelated operations. This means that the method has low cohesion which makes it difficult to reuse. Long parameter list means that a method is taking too many parameters. Long parameter lists are difficult to understand and they are continuously changing, as the data needed by the method is varying. Feature envy means that a method is more interested in other classes than the class it is currently located in. A method with the Feature Envy smell should be moved to a class that the method is mainly operating with.

Experiment B. In the second experiment no predetermined viewpoints for subjective evolvability evaluations were given, i.e. the existence of the smells was not asked. Instead the evaluators were asked only if the method was in such a state that it should be refactored.

3.2.3. Evaluators. The evaluators of both experiments were students of the Software Testing and Quality Assurance course at Helsinki University of Technology (HUT) in fall semesters 2003 and 2004. Thus, both experiments had a unique set of similar evaluators. The evaluators of both experiments are introduced below. Table 1 summarizes the most important demographic variables and shows that the evaluators in the experiments were similar.

Experiment A. In fall 2003, the course had 82 students, and 51 of them participated in the experiment. We rewarded the participants by extra points. Five outlier evaluators were removed from these 51 students based on their answers, e.g. one evaluator acclaimed that there was a lot of Long Parameter List smell when

the method had no parameters.

The evaluators were studying for a M.Sc. degree that requires a minimum of 180 credits. The study times at HUT fluctuate greatly. Thus, we measured the number of credits rather than number of years studied. On average the students had 115 credits indicating they had completed approximately two thirds of their studies. 89,1% (41/46) of the students had between 70 and 158 credits.

Many HUT students also work in the software industry during their studies. Therefore, the work experience in software development was asked. 13 evaluators (28,3%) had no software development work experience, but 27 (58,7%) of the evaluators had a year or more work experience in software development.

Other demographic data collected was: number of years studied, department and main subject of study, subjective evaluation of their knowledge of Java and UML, subjective evaluation of programming ability against other HUT students, and the perceived importance of good program structure.

Experiment B. In fall 2004 37 students participated in the experiment. The students were given extra points based on the quality of the rationales they provided in the experiment. One outlier student was removed based on the non-sense rationales given.

The evaluators had an average of 124 credits. 86,1% (31/36) of the students had between 80 and 160 credits. The mean programming related work experience was 1,8 years. Eight students (22,2%) had no work experience, but 21 students (58,3%) had a year or more programming work experience.

Additional demographics collected were: grades of programming courses, subjective evaluation of their knowledge of Java and UML, subjective evaluation of programming ability against other HUT students, experience in software maintenance, and the perceived importance of good program structure.

Table 1 Evaluators of the experiments

	Experiment A	Experiment B
N	46	36
One year or more software development work experience	58,7%	58,3%
Credits (mean (std. dev) / min-max)	115 (31,2) / 34-187,5	125 (38,3) / 6-199
Years of work experience (mean (std. dev) / min-max)	1,6 (1,76) / 1 / 0-7	1,9 (2,91) / 1 / 0-15

3.2.4. Source code metrics. We selected source code metrics to analyze the software elements based on suitability to measure the studied smells and their recognition in the literature. From size metrics we calculated Lines of Code (LOC), Number of Parameters (Par) and Cyclomatic Complexity (CC). We measured coupling with the Number of Remote Methods called (NR) and the number of couplings between a method and objects (CBO). Finally, we measured fan-out (FO) that is the number of reference types used in formal parameters, throws declaration, and local variables. The results are in Table 2.

Table 2. Code metrics of the methods

Method	Metrics					
	LOC	Par	CC	CBO	NR	FO
DiskManager.readFromDisk	67	1	11	5	16	4
DiskManager.writeToDisk	48	1	7	6	23	7
FamilyFrame.addRelationClicked	20	3	6	4	6	1
FamilyFrame.FamilyFrame	84	0	1	13	17	3
Person.dateOfBirthEquals	24	1	2	1	3	1
Person.getChildren	9	0	3	4	7	3
Person.illegalRelation	46	1	12	5	8	3
PersonTableModel. applyChangesToPerson	11	4	1	2	5	1
PersonTableModel.personMatch	19	6	1	1	6	2
PersonTableModel.searchPersons	21	5	3	2	5	4

3.2.5. Experiment material and evaluation session. Experiment A. The responses were collected with a survey form containing the methods under evaluation. All the survey forms were unique as the methods were placed in random order for each. Additionally, descriptions of the smells and a UML-diagram of the software were handed out.

The evaluators evaluated how much of each smell existed in each of the methods. The question was *Do these smells exist in the method below?* and it was evaluated with a seven point ordinal scale with one standing for *Not at all* and seven standing for *Yes very much*. A question about refactoring was also presented; *Would you refactor the method to remove the smells (in order to keep the software easy to understand and develop further).* This was answered with the following five point ordinal scale: *1-No, 2-Unlikely, 3-Maybe in the future, 4-Yes if the method needs further development, 5-Yes, immediately.*

The experiment was run as a single lecture session, which took altogether about 70 minutes. It began with a 20-minute introduction lecture that covered general information about the subjective evaluations, ideas why good software structure is important, the exercise organization, and an explanation of the smells under evaluation. To guarantee that all the evaluators used the same amount of effort in evaluating each method, the evaluation time was restricted to five minutes, i.e. the evaluators were not allowed to proceed until they were instructed to do so.

Experiment B. Experiment B was entirely web

based. The instructions of the experiment consisted of the following: the task, explanation of refactoring and its benefits, the grading of the assignment, the estimated effort required, the description of the evaluated software application with screen shots, the UML class diagram, the source code, and the executable application. First, there was a set of demographic questions. Then the methods were presented with the question *Would you refactor the method <in question> in order to keep the software easy to understand and develop further?* For the answers there were five options *1-No, 2-Unlikely, 3-Maybe 4-Yes, later when the method needs further development, 5-Yes, immediately*. In connection with the questions there was a hyperlink to the source code of the method. The rationale for the evaluation was asked using a question: *Explain your choice? If refactoring is needed, explain what and how the method should be refactored. If the method is OK, explain what desirable qualities the method possesses. If you answered Maybe also give your rationale*.

Experiment B did not have a lecture before the evaluators made their evaluations. The evaluators had received information about the software to be studied and the benefits of refactoring from a web page. Experiment B did not ask the evaluators to search for smells or any other evolvability flaws in the software. Each evaluator participated in the experiment through a web-based survey, where they were able to browse back and forth while answering. The time spent to complete the survey was tracked by the web-based survey system.

3.3. Data analysis

This section presents the methods used to analyze the interrater agreement and the factors explaining the evaluations.

3.3.1. Interrater agreement. The Kendall coefficient of concordance [9] which is referred to as *W* or Kendall's *W,* can be used to study the agreement between three or more raters on several related samples. Other measures such as Kappa and Kendall's Tau, can measure interrater agreement only between two raters and, therefore, were not applicable.

Kendall's *W* tells the amount of interrater agreement by a number between 0 and 1. If all raters agree *W*=1 and *W*=0 implicates no agreement. Additionally, the statistical significance of the agreement was studied. High significance (p-value < 0.01) means at least partial concordance among the raters.

Interrater agreement can also be studied in other fields like ski-jumping. For a reference Kendall's *W* was calculated from the first round results of the ski jumping world cup competition held in Oberstdorf, Germany at 29th December 2004. On that occasion the 5 judges evaluating 50 jumps achieved Kendall's *W* 0,888 and asymptotic significance p-value 0,000.

3.3.2. Factors explaining evaluations. Regression analysis was used to study how the method characteristics and the evaluators' demographics affected the evaluations. The data was in various scales (nominal, ordinal, interval), and therefore could not be analyzed using classical linear regression. The data was analyzed using categorical regression that is available in the SPSS™ software. Categorical regression is founded on optimal scaling, which turns nominal and ordinal variables into linear variables [12].

Several regression models were created for each of the predicted (dependent) variables that were the evaluations of the existence of three code smells and the refactoring decision. The first model (called the MetDem model) used the source code metrics of the methods and the demographics of the evaluators as predictor (independent) variables. The second model contained only the demographics and the third model only the source code metrics of the method. In Experiment A an additional model was constructed that used the smell evaluations as predictor variables when predicting the refactoring decision.

4. Results

This section shows the results of the study. Discussion of the results and answers to the research questions are presented in Section 5.

4.1. Interrater agreement

Table 3 shows the results of the interrater agreement analysis. The evaluators had a high agreement on evaluations concerning the Long Method and Long Parameter List smells. The agreements concerning the Feature Envy smell and the refactoring decisions are considerably weaker. However, all *W* values are significant indicating that the evaluators had at least some level of agreement. *W* values of the refactoring decision for both experiments are close to each other. The number of evaluations of evaluators varied from 44 to 46 in Experiment A. In Experiment B there were 36 evaluators' evaluations. For all the cases the number of evaluated objects was 10.

Table 3. Interrater agreement

Question	N	*W*	Sig.
Exp A – Long Method	46	0,777	0,000
Exp A – Long Parameter List	46	0,816	0,000
Exp A – Feature Envy	44	0,238	0,000
Exp A – Refactoring	45	0,353	0,000
Exp B – Refactoring	36	0,397	0,000

4.2. Explaining factors - regression analysis

This section studies the factors explaining the evolvability evaluations.

4.2.1. Long Method. Regression models for the Long Method smell are in Table 4. From the table, we can see that the MetDem model, consisting of the source code metrics and evaluators' demographics, explains 74,6% of the evaluations. The MetDem model details revealed that the source code metrics were the most important predictors. Also the model with just the metrics, namely the Metric model, predicts 71,2% of the evaluations. However, the Demographic model, containing the information about the evaluators' background, is not able to effectively explain the evaluations.

Table 4. Long Method regression models

Model	Adj. R Square	Sig.
MetDem	0,746	0,000
Metric	0,712	0,000
Demographic	0,012	0,231

As it seems that the metrics rather than the demographics explained most of the Long Method evaluations it made sense to study them in more detail. In Table 5 we can see the predictor variables of the Metric model and their standardized betas, significance level of the F-values, and the correlation with the predicted variable.

Table 5. Predictors in the Metric model for the Long Method smell evaluations

Predictor	Std. β	Sig.	Correl.
Par	-0,108	0,002	-0,509
LOC	0,738	0,000	0,815
CC	0,114	0,001	0,474
FO	0,008	0,866	0,480
NR	0,171	0,014	0,700
CBO	-0,190	0,001	0,624

Table 5 shows Lines of Code as the most important predictor in the model. The Lines of Code in this case meant that a single line is a single line of code regardless of the space usage or comments in the method. NLOC[1] was also tested in the regression model, but it performed slightly poorer, although the difference was marginal. Other metrics had only a subsidiary effect in the Metric model, but most of them had a high correlation with the Long Method evaluations. This indicates that a reasonably good regression model could be created even without the lines of code metric, and when this was tested the Metric model without Lines of Code was able to explain 61,8% of the evaluations. In that model number of remote methods (std. beta 0,531), cyclomatic complexity (std. beta 0,335), and coupling between objects (std. beta 0,255) were the best predictors.

[1] lines of code without comments and blank lines

4.2.2. Long Parameter List. The regression models for the Long Parameter List smell are in Table 6. The results are similar to the results of the Long Method smells. The MetDem model explained 77,6% of the evaluations, and the Metric model was almost as good explaining 76,1% of the evaluations.

Table 6. Long Parameter List regression

Model	Adj. R Square	Sig.
MetDem	0,776	0,000
Metric	0,761	0,000
Demographic	0,054	0,003

Further analysis of the Metric model in Table 7 shows that the Number of Parameters is the most important predictor. It has very high correlation with the predicted variable. Number of Remote Methods and Fan Out also have some impact in the regression model. However, it seems likely that the effect was caused more by the limited amount of methods evaluated rather than by real effect. Additionally, only the Number of Parameters metric has a positive correlation with the predicted variable.

Table 7. Predictors in the Metric model for Long Parameter List smell evaluations

Predictor	Std. β	Sig.	Correl.
Par	0,807	0,000	0,857
LOC	0,095	0,074	-0,466
CC	-0,170	0,000	-0,424
FO	0,229	0,000	-0,228
NR	-0,287	0,000	-0,441
CBO	0,061	0,246	-0,494

4.2.3. Feature Envy. Regression models for the Feature Envy smell are in Table 8. The MetDem model was able to explain only 29,8% of the evaluations. The Metric Model explained only 9,8% of the evaluations. Thus, with Feature Envy it appears that the predictors failed in predicting the Feature Envy smell evaluations. Consequently, there is no need to look at the individual models any further.

Table 8. Feature Envy regression models

Model	Adj. R Square	Sig.
MetDem	0,298	0,000
Metric	0,098	0,000
Demographic	0,054	0,003

4.2.4. Refactoring decision. In refactoring decision regression analysis there were data from both experiments. The regression models of the refactoring decision can be seen in Table 9 and Table 10. In Table 9 we can see that the SmeMetDem model which consists

of smell evaluations, source code metrics and demographics, explained 66,5% of the evaluations. The most significant contributors in the SmeMetDem model are the smell evaluations. The Metric model explained 31,9% of the evaluations. Finally the Demographic model was not effective, explaining only 8,7% of the evaluations.

Table 9. Refactoring decision regression models in Experiment A

Model	Adj. R Square	Sig.
Exp A – SmeMetDem	0,665	0,000
Exp A – Smell	0,618	0,000
Exp A – MetDem	0,435	0,000
Exp A – Metric	0,319	0,000
Exp A – Demographic	0,087	0,000

Experiment B did not have a SmeMetDem or Smell model because no questions concerning the code smells were asked in that experiment. As can be seen from Table 10, the MetDem model explained 28,4% of the evaluations. Further analysis showed that most of the explanative power came from the source code metrics. The Metric model explained 26,1% of the evaluations alone while the demographic model failed to be effective. The comparison to Experiment A shows that the Demographic model in Experiment A performed slightly better, but this is likely to be caused by the larger set of demographic variables in Experiment A.

Table 10. Refactoring decision regression models in Experiment B

Model	Adj. R Square	Sig.
Exp B – MetDem	0,284	0,000
Exp B – Metric	0,261	0,000
Exp B – Demographic	0,036	0,026

Details of the Smell Model from Experiment A can be seen in Table 11. From the table we can see that all the smell evaluations were important when predicting the refactoring decision. The Long method smell evaluations were the most considerable contributors, but even the evaluations of the Feature Envy smell contributed significantly to the Smell model.

Table 11. Predictors of the Refactoring decision in Smell Model in Experiment A

Predictor	Std. β	Sig.	Correl.
Long Method	0,598	0,000	0,514
Long Parameter List	0,469	0,000	0,280
Feature Envy	0,360	0,000	0,518

In Table 12 we can see the predictors of the Metric model predicting the refactoring decision in both experiments. The lines of code measure had the highest beta in both regression equations, and it also had the highest correlations with the refactoring decision. In the regression models, Coupling between objects (CBO) is a suppressor term, which traditionally would indicate a reduction in the likelihood of refactoring. However, this is more likely caused by the multicollinearity that source code metrics have with each other. This is supported by the facts that CBO had positive correlation with the refactoring decision and that Lines of Code and CBO also had high correlation with each other (Person correlation 0,830 p-value 0,000). This indicates that in fact increase in CBO does not decrease the refactoring need.

Table 12. Predictors of the Refactoring decision in Metric model in Experiments A and B

Predictor	Experiment A			Experiment B		
	Std. β	Sig.	Correl.	Std. β	Sig.	Correl.
Par	0,397	0,000	0,130	-0,133	0,032	-0,230
LOC	0,805	0,000	0,381	0,923	0,000	0,453
CC	-0,003	0,957	0,157	-0,073	0,230	0,233
FO	0,063	0,417	0,215	0,057	0,534	0,168
NR	0,011	0,916	0,157	-0,162	0,197	0,277
CBO	-0,299	0,001	0,272	-0,500	0,000	0,246

5. Discussion

This section provides the discussion where we first examine the answers to the research questions. Second, the limitations of the study are addressed.

5.1. Answers to the research questions

Research Question 1: *Is there an interrater agreement in subjective evolvability evaluation?* This research question was studied in Section 4.1. Kendall's coefficient of concordance (Kendall's *W*) was used to measure the agreement between evaluators.

In the evolvability evaluations we saw that code smells Long Method and Long Parameter List produced a high agreement between raters having Kendall's *W* of 0,777 and 0,816 respectively. The high agreement on these smells is not surprising, since both of them should be easy to evaluate and rank. Long Parameter List can be clearly seen by looking at how many parameters are passed to the method. Long Method could be a little more difficult since the definition given told that such methods have low cohesion, are long, and difficult to understand and reuse. Still, the evaluators had very high agreement on the Long Method smell.

The Feature Envy smell had the lowest coefficient of concordance with 0,238. However, from the feedback of the experiment we learned that some evaluators (3/46) felt that they did not completely understand what was meant by the Feature Envy smell. This can partly explain the low interrater agreement in this case.

The Kendall's *W* of the refactoring decision in Experiment A was 0,353 and in Experiment B 0,397. This was considerably lower than on code smells Long

Method and Long Parameter List. *W* values in the refactoring question in both experiments were very close to each other (only a difference of 0,044). This indicates that the level of interrater agreement is not affected by the different setups in the experiments. However, one might expect the agreement on the refactoring decision to be higher in Experiment A where the evaluators had the smell descriptions available to help them in making the refactoring decision.

The refactoring question really is the key of the experiments because it asked if the method is in such condition that it should be improved to make it more evolvable. The result seems to indicate that there are differences in people's opinions on whether a certain piece of code should be refactored or not.

Based on the data the answer to research question 1 is two-folded. For simple code smells Long Method and Long Parameter List there was a high agreement between the evaluators. For the refactoring decision and the Feature Envy smell the level of agreement was considerably lower. Since all evaluations are significant we must conclude that there is partial concordance among the evaluators in all evaluations. However, the level of agreement is not satisfactory in all cases.

Comparison to prior work [7, 8, 14-16] is challenging due to lack of proper representation of the evaluation data [13, 14, 16], lack of statistical power [8, 10], and use of non-standard statistical methods[2] [10, 14, 15]. All prior work lacks calculation of statistical significance on the interrater agreement and Kendall's *W* making it impossible to tell whether the raters really were in agreement on the evaluated software or not.

Research Question 2: *How much of the evolvability evaluation of a software element can be explained by the measurable characteristics of the software element and the evaluator demographics?* This research question was studied in Section 4.2. This research question was studied using categorical regression founded on optimal scaling making it possible to use continuous and non-continuous variables as both dependent and independent variables.

We saw that the evaluations on code smells Long Method and Long Parameter List could be predicted with good accuracy by the regression models. In the Long Method smell 71,2% of the evaluations could be explained by the regression model consisting of source code metrics. As expected, the lines of code metric was the most important predictor for the Long Method evaluations. However, the Metric model without the lines of code metric explained 61,8% of the evaluations. This is caused by the correlation the source code metrics have with each other. In Long Parameter List evaluations the Metric model explained 76,1% of the evaluations. The most important predictor in the model was the number of parameters, and unlike in the Long Method evaluations there was no substitute for this predictor which is not surprising.

The explanation power of the code metric based regression models diminished when we studied the refactoring decision and the Feature Envy code smell evaluations. The source code metrics explained only 9,8% of the Feature Envy smell evaluations. For the refactoring decision the percentage of the evaluations explained by the Metric models was 31,9% in Experiment A and 26,1% in Experiment B. In Experiment A we were able to use the smell evaluations of each evaluator to create another regression model that explained 61,8% of the refactoring decision.

We also studied demographic variables as predictors for a refactoring decision, but their explanatory power was low. We even tried to improve the gathering of demographic data in Experiment B grounded on Experiment A, but still the background variables had only minor explanatory power. In fact the demographic variables performed slightly better in Experiment A, but this was likely affected by having more demographics variables in Experiment A rather than a real improvement in the data variables.

Comparing these results to the results of research question 1 reveals us that both the interrater agreement and the metric regression models have a similar two-folded structure. Both perform well on Long Method and Long Parameter List smell evaluations. Similarly both the interrater agreement and the regression models have low values when it comes to the refactoring decisions and especially Feature Envy evaluations. There is a connection between these two, and it is caused by the fact that the source code metrics of any method will remain the same even if there is disagreement between raters. Thus, if there is disagreement whether a certain method should be refactored, it automatically means that the code metrics of that method cannot make up a strong regression model that would predict the refactoring decision because there is a disagreement on the issue. Naturally this does not affect the regression model created from the smell evaluations because the possible disagreement in the refactoring decision is likely to be reflected in the smell evaluations.

Comparison to prior studies is not clear cut because they have not utilized regression analysis. Regardless, we can try to make some comparisons. Kafura and Reddy [7] concluded that the expert evaluations on evolvability were in conformance with the complexity source code metric. This conflicts with our results since we showed that metrics were not sufficient predictors of the refactoring decision i.e. the evolvability improvement need. However, their results are based

[2] The researchers have calculated the percentage of answers that were off by n steps in their ordinal scale, or they calculated averages and standard deviations from the ordinal scale..

on interviews on software maintainability while we used the refactoring decisions on an ordinal scale survey. Oman et al. [13, 16] used subjective evaluations to create a metrics based maintainability measure. It is therefore quite natural that their metrics correlated well with subjective evaluations.

Our prior work [10], which studied code smell evaluations at the module level in an industrial setting, concluded that source code metrics and code smell evaluations did not correlate. However, this study shows that simple code smells and source code metrics have a relationship. The difference is likely to stem from three limitations in our prior study. First, in our prior work we used higher (module) level evaluations. Second, the evaluations were based on recollection. Third, the evaluators had been working with the software modules and, thus, had a more personal bond with the software under evaluation. We think that these dissimilarities can explain the different results.

5.2. Limitations

This section assesses the limitations of the study. Threats to internal and external validity are studied based on [1, 2], and improvements to the experimental design are proposed.

5.2.1. Threats to internal validity. In Experiment B the reliability of the respondents' procedures was questionable since it was done as a web survey, and there was no control over the respondents. However, a similar situation could have occurred in Experiment A, because we had no means to make sure that the evaluators actually paid attention to the introduction lecture they were given prior to the experiment.

Experiment B lacked the randomization of the methods evaluated. The evaluation order of the methods could have caused bias to the evaluation results. In Experiment B the evaluators had the possibility to go back and forth in their answers if they wanted to check or change something in their prior answers. This should have limited the effect cause by the lack of randomization.

5.2.2. Threats to external validity. There could be interaction between the selection of evaluators and the issue studied in the experiments. In both experiments the population was the students participating in the course. However, proper sampling was not done to select the individuals who would become the evaluators of the experiment. Instead, the evaluators were those interested in receiving the extra credit for their course grade. This sampling method could have caused bias. It is possible that students who were more interested in this topic participated in the experiment. Based on the course grades there actually was a slight bias in both experiments towards better performing students.

Another threat to external validity comes from the population. Generalizing the results obtained using students to developers in industry might not be possible. We may argue that students are a too homogenous group, and, therefore, the results are too good. Furthermore, one may argue that as the evaluators had a varying amount of industrial programming experience (from zero to fifteen years), the population is too heterogeneous. However, also teams of industrial developers can have fluctuating levels of homogeneity.

The selection of the evaluated software elements is another threat to external validity. It is possible that with a different set of software elements different results could be obtained.

The source code metrics used to measure the evaluated methods were limited to six different metrics, which were introduced in Section 3.2.4. With a different set of metrics the results could have been different. However, it must be pointed out that the goal was not to discover the best metrics to predict refactoring decisions, but to test how a few widely used measures perform in predicting the evaluations.

It is possible that the results represent more the effect of the experimental setting than what it would be in the real world. One may argue that in the real world the agreement between raters would be better since the raters would understand and study the evaluated piece of software longer and more thoroughly.

5.2.3. Improvements to the experiment design. This section provides a brief list of how the experiment should be improved.

1. The selection of the evaluators should be random.
2. The population should be more diverse.
3. More software elements should be used for evaluation to reduce the possible bias.
4. More source code metrics not suffering from multicollinearity should be measured.
5. The list of code smells should be longer to allow better comparison between the evaluations.
6. There should be several questions describing each smell to allow assessing reliability of answers.
7. A pre-exam should be held to see how well the evaluators understand the issues to be evaluated.

6. Conclusions and future work

This paper investigates subjective evaluation of software evolvability by assessing interrater agreement and factors explaining the evaluations. We have seen that the interrater agreement is high on the simple code smells Long Method and Long Parameter List, but considerably lower on the refactoring decision and the Feature Envy code smell. Regression models based on

source code metrics explained over 70% of the evaluations of Long Method and Long Parameter List smells, but explained only about 30% of the refactoring decision. The best predictors of the refactoring decision were the evaluations of the code smells explaining more than 60% of the decisions.

High interrater agreement on the simple code smells implies reliability of the smell evaluations. Thus, there should be no need to double check the evaluations of those smells by tools or another developer. Additionally, for the simple code smells the prediction by the code metrics based regression model was also quite accurate. This suggests code metrics tools usage as an effective approach in highlighting straightforward problems in the code.

Lower interrater agreement of the refactoring decision indicates a possible unreliability in the developers' evaluations. To compensate this it seems advisable to double check the evaluation at least from time to time because false judgments may lead to poorly evolvable software. The lower interrater agreement also raises a topic to be addressed in our future work concerning the causes explaining the differences in the refactoring decision evaluations. Additionally, metrics-based regression models were only able to explain 30% of the refactoring decisions. This indicates difficulties in building tool support to simulate real-life subjective refactoring decisions.

In the future we will analyze the rationales provided by the evaluators in Experiment B. This allows us to study the qualitative elements, i.e. the real reasons behind the refactoring decisions and the proposed changes to the source code. In the future one could also study the effect of coding standards and team context to evolvability evaluations. Presumably, a team with a coding standard should have higher interrater agreement compared to a group of students. Finally, a path that should be pursued is the investigation of using machine learning techniques to identify poorly evolvable software according to evolvability evaluations. However, for this to be feasible the interrater agreement must be sufficient and we must also understand the rationales behind the evolvability evaluations.

References

[1] D.T. Campbell and J.C. Stanley, *Experimental and quasi-experimental design for research,* Chicago, USA: Rand Mcnally College Publishing Company, 1966.

[2] T.D. Cook and D.T. Campbell, *Quasi-experimentation: Design and analysis issues for field settings,* Chicago, USA: Rand Mcnally College Publishing Company, 1979.

[3] M.A. Cusumano and R.W. Selby, *Microsoft Secrets,* USA: The Free Press, 1995.

[4] M. Fowler and K. Beck, "Bad Smells in Code," in *Refactoring: Improving the Design of Existing Code,* 1st ed., Boston: Addison-Wesley, 2000, pp. 75-88.

[5] M. Genero, M. Piatini and E. Manso, "Finding "early" indicators of UML class diagrams understandability and modifiability," *in Proceedings of International Symposium on Empirical Software Engineering,* 2004, pp. 207-216.

[6] M. Genero, M. Piattini and C. Calero, "Empirical validation of class diagram metrics," *in Proceedings of the International Symposium on Empirical Software Engineering,* 2002, pp. 195-203.

[7] D.G. Kafura and G.R. Reddy, "The Use of Software Complexity Metrics in Software Maintenance," *IEEE Trans. Software Eng.,* vol. 13, no. 3, 1987, pp. 335-343.

[8] Y. Kataoka, T. Imai, H. Andou and T. Fukaya, "A Quantative Evaluation of Maintainability Enhancement by Refactoring," *in Proceedings of the International Conference on Software Maintenance,* 2002, pp. 576-585.

[9] M. Kendall Sir, "The problem of *m* ranking," in *Rank Correlation Methods,* 5th ed., J.D. Gibbons Ed. London: Edward Arnold, 1948, pp. 117-143.

[10] M.V. Mäntylä, J. Vanhanen and C. Lassenius, "Bad smells - Humans as code critics," *in Proceedings.20th IEEE International Conference on Software Maintenance, 2004.* 2004, pp. 399-408.

[11] T. Mens and T. Tourwe, "A survey of software refactoring," *IEEE Trans. Software Eng.,* vol. 30, no. 2, 2004, pp. 126-139.

[12] J.J. Meulman, "Optimal scaling methods for multivariate categorical data analysis," SPSS., Tech. Rep. SPSS White Paper, 1998.

[13] P.W. Oman and J. Hagemeister, "Constructing and testing of polynomials predicting software maintainability," *Journal of Systems and Software,* vol. 24, no. 3, 1994, pp. 251-266.

[14] M.J. Shepperd, "System architecture metrics for controlling software maintainability," *in IEE Colloquium on Software Metrics,* 1990, pp. 4/1-4/3.

[15] B. Shneiderman, *Software Psychology: Human factors in Computer and Information Systems,* Cambridge, Massachusetts, USA: Winthrop Publishers, 1980.

[16] K.D. Welker, P.W. Oman and G.G. Atkinson, "Development and application of an automated source code maintainability index," *Journal of Software Maintenance: Research and Practice,* vol. 9, no. 3, 1997, pp. 127-159.