Automated use case similarity computation can aid the assessment cohesion and method complexity of classes

Renato C. Juliano, Bruno A. N. Travençolo, Michel dos S. Soares, Marcelo de A. Maia
Computer Science Department
Federal University of Uberlândia
Uberlândia, Minas Gerais, Brasil
travencolo@gmail.com, michel@facom.ufu.br, marcmmaia@facom.ufu.br

Institute of Exact and Human Sciences
Centro Universitário do Planalto de Araxá
Araxá, Minas Gerais, Brasil
renatocorrea@uniaraxa.edu.br

Abstract—Use cases are widely used in early software development phases such requirements analysis. In this paper, we investigate how use case similarity could impact the classes that implement them. We studied if the similarity of use cases could have impact on the lines of code shared between them and on metrics of classes that implement them, such as, coupling, cohesion and method complexity of classes. We also have successfully applied an automated approach to assess the similarity of use case names. We found that there is a statistically significant correlation, although not strong, between use case similarity and sharing of lines of code. Interestingly, we also have found that classes that are shared between different use cases tend to have lower cohesion. Moreover, classes that are shared between similar use cases tend to have higher method complexity in classes. We found no relation between use case similarity and coupling.

Keywords—Use Case Similarity, Software Metrics, Software Maintenance

I. INTRODUCTION

Software maintenance has been object of study for many years [2], [10], [16], as it is well-established that software needs to evolve in order to keep being useful. Software maintenance accounts for up to 80% of software costs [8], [18]. In addition, according to some studies, the faster an error is discovered, the easier and less expensive and time-consuming is to correct it [6]. Therefore, any activity that facilitates the discovering of software errors or at least helps in predicting future problems is useful for maintaining the software. It is even more appropriate if this activity belongs to the initial phases of the software development process, such as during requirements analysis and design activities.

Use Cases, which are commonly described using both a tabular approach or simple natural language and using the UML Use Case diagram, are responsible for describing the external activities of a software [11]. Use cases are useful to describe the functions to be provided by the software, usually from an actor point of view.

The proposal in this article is to describe a relationship between the software metrics CBO, LCOM and WMC, proposed in 1994 by Chidamber and Kemerer [3], with the level of similarity between the items of a Use Case diagram. The process to do this consists of two steps.

In the first step, we will study if it is feasible to use an automated algorithm to assess use case similarity based on algorithms for computation of string similarity. This step will be evaluated by human experts. In the second step, we will study how the similarity between two use cases can be related to CBO, LCOM and WMC of the concrete software.

The reminder of the article is organized as follows. The next section is about describing the literature basics to perform this work. Among the topics, we described the analysis of similarity between strings, software metrics. In section III, we describe the approach to collect data, to extract software metrics, and to relate software metrics with the level of similarity between use cases. In Section V the results of the article are described, and Sections VI and VII brings the threats to validity and the conclusion.

II. BACKGROUND

A. Similarity between texts in use case

In order to perform the analysis proposed in this paper, it is necessary to find similarities between the items in a use case diagram. There are two ways in doing this analysis, one by using a human specialist and other by automated techniques.

Human analysis has a series of advantages, specially regarding to the context awareness and interpretation of the texts, combined with the experience of the developer. However, these advantages are only valid when the experience and the knowledge about the problem are clearly defined, otherwise the recognition will be compromised.

Automated computation of similarity can significantly reduce the time of the analysis. However, this is not a trivial task, mainly because to the difficult of defining the context of the used words [5]. Several proposal can be find in the literature
in order to evaluate similarity between texts [4]. In this work we use the Jaro-Winkler [22] and Levenshtein [12] proposals.

In this work, we found a similarity threshold value that balanced the precision/recall relation according to the analysis of human specialists. Therefore, it is possible to automate the process of similarity computation.

B. Software Metrics

Many studies have shown that the process of software development can be significantly improved if the quantification of involved process are adopted along all stages [3]. Several metrics were proposed in order to quantify different features of the software [1], [3], [13]. Among then, in this work we used the set of metrics proposed by Chidamber e Kemerer [3]. This set of metrics, known as CK suite, is composed by six metrics related to the object-oriented paradigm [9].

The use of CK metrics can indicate several characteristics of the process of software development, as error-proneness [15], [19], [21], bad smells identification [17], analysis of the effectiveness of refactoring [20] and assessment of the maintainability of the process of software development [6].

In this work, three CK metrics were used, namely WMC, CBO and LCOM:

- WMC - Weighted Methods Per Class: The WMC is a measure defined by the sum of the complexity of the methods of the class. The definition of the complexity was not specified by Chidamber e Kemerer [9], which suggested the use of a measure expressed by a natural number. The WMC is related to the number of methods of the class, and their complexity may be proportional to the amount of time necessary to develop and maintain the class (the greater the number of methods of a class, the greater the impact on their subclasses). Classes with a large number of methods normally refer to specific applications, reducing the possibility of reuse [3]. In this paper, as WMC, will be used McCabe Cyclomatic Complexity [14].

- CBO - Coupling between object classes: The CBO metric represents the number of classes in which it is coupled. One class is coupled to another when at least one method of one of the classes uses the method or attributes of the other class. By using this measurement it is possible to identify the level of reuse of a class and its degree of modularity. The lower is the value of CBO, more modularized is the class and, as consequence, the possibility of reuse increases. In addition, it is possible to analyse the rigor of software testing, by means of the degree of importance, the amount of time used and number of tests. As higher is the CBO, more complex will be the testing [9].

- LCOM - Lack of Cohesion in Methods: The LCOM metric quantifies the similarity between the methods inside a class. Two or more methods are cohesive when they share the attributes of the class. When the value of LCOM is high, it means that the class does not have a well defined functionality, as several methods modify the same attributes. On the other hand, if LCOM is low, the methods of the class are cohesive and, hence, less similar [9].

III. STUDY SETTING

The conduction of this study was organized in several steps that will be described below.

Definition of the subject system

The system used in this study was a system in the educational domain. The system has 17 use cases, which are implemented with 3451 LOC, 34 classes, 181 methods. The system is written in C#. There are several reasons that justified the choice of the system: it is a real working system with over 20 end-users, it has well-documented use cases, the company that developed the system has provided both the source code and developers, which were allocated to recover the traceability of use cases and classes that implement them and to assess the results of the similarity algorithms.

Use cases

The use cases of the subject system used in this study is shown in Table I. We decided to maintain the original names in Portuguese (the language used in the specification), because this was the actual input to similarity computation. Moreover, the reader can still intuitively see the possibilities of similarity.

<table>
<thead>
<tr>
<th>UserCaseID</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>uc1</td>
<td>Lancar Dados da Banca</td>
</tr>
<tr>
<td>uc2</td>
<td>Visualizar TCC Postados</td>
</tr>
<tr>
<td>uc3</td>
<td>Disponibilizar Visualizacao do TCC no Site do Uniaraxa</td>
</tr>
<tr>
<td>uc4</td>
<td>Manter titulo do TCC</td>
</tr>
<tr>
<td>uc5</td>
<td>Impressao da ato de defesa</td>
</tr>
<tr>
<td>uc6</td>
<td>Impressao da lista de presenca</td>
</tr>
<tr>
<td>uc7</td>
<td>Impressao da lista de alteracoes/correcoes</td>
</tr>
<tr>
<td>uc8</td>
<td>Lancar nota/status do TCC</td>
</tr>
<tr>
<td>uc9</td>
<td>Escolha dos alunos que fariam TCC</td>
</tr>
<tr>
<td>uc10</td>
<td>Postar TCC</td>
</tr>
<tr>
<td>uc11</td>
<td>Incluir titulo do TCC</td>
</tr>
<tr>
<td>uc12</td>
<td>Visualizar nota/status do TCC</td>
</tr>
<tr>
<td>uc13</td>
<td>Lancamento do Status TCC para a disciplina</td>
</tr>
<tr>
<td>uc14</td>
<td>Relatorio Grafico por curso: alunos aprovados vs todos</td>
</tr>
<tr>
<td>uc15</td>
<td>Relatorio de alunos aprovados/reprovados por curso</td>
</tr>
<tr>
<td>uc16</td>
<td>Lancamento da disciplina de TCC</td>
</tr>
<tr>
<td>uc17</td>
<td>Visualizacao do titulo do TCC</td>
</tr>
</tbody>
</table>

Traceability recovery of use cases and classes

This step was conducted with the team that participated actively in the development and documentation of the system. The team consisted of four developers with at least three years of experience. The result of this step was the definition of a set of classes that implemented each use case. It is possible that a class participates in the implementation of one or more use cases.

Evaluation of similarity algorithms

This step was conducted to evaluate if algorithms to evaluate string similarity would be a substitute for human analysis of use case similarity. The approach consisted in applying an algorithm to compute the string similarity, which was tested in a pairwise comparison of all pairs of use cases. The chosen algorithms was Jaro-Winkler [22] and Levenshtein [12]. The results of the algorithms is a number between zero (completely dissimilar) and one (completely similar). The result of the algorithm was assessed against a gold answer set.
which was produced by the development team consisted of four developers with two, seven, eight and ten years of experience, respectively. The developers received the use case diagram of the system and the following instructions to perform the comparison:

- Consider similarity use case names (titles) that have some semantic similarity between them that could indicate the sharing of implementation classes.
- Select the pairs of use cases that you consider that have some degree of similarity.

After the instruction phase, the developers produced their set with pairs of similar uses cases. After that, they shared and discussed their results to reach a consensus.

In order to define a threshold for the similarity algorithm we used a ROC curve to assess precision and recall values of the similarity algorithms. The area under the ROC curve of Jaro-Winkler was higher than the area of Levenshtein, so we decided to consider the algorithm Jaro-Winkler. Figure 1 shows the similarity values for use case pairs that were considered similar by the developers (right) and the similarity values of the pairs that were not considered similar by developers (left). This figure shows the feasibility of automatic similarity computation.

The next step was the definition of the threshold for balanced precision/recall of the similarity algorithm. Table II shows an inflexion point in the ROC curve that provide a balanced threshold. For that threshold (similarity > 0.6409), we get recall equals 60% and precision equals 87.93%.

### TABLE II. PRECISION AND RECALL OF JARO-WINKLER BASED ON EVALUATION BY DEVELOPERS

<table>
<thead>
<tr>
<th>Feature</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Similarity</td>
<td>&gt; 0.6409</td>
</tr>
<tr>
<td>Recall</td>
<td>60.00</td>
</tr>
<tr>
<td>95% CI</td>
<td>36.05% to 80.88%</td>
</tr>
<tr>
<td>Precision</td>
<td>87.93</td>
</tr>
<tr>
<td>95% CI</td>
<td>80.58% to 93.24%</td>
</tr>
<tr>
<td>Likelihood</td>
<td>4.97</td>
</tr>
</tbody>
</table>

**Classification of classes shared between use cases**

In our study, our goal is to analyze if classes shared between similar and non-similar use cases have special characteristic concerning the metric values for LCOM, CBO and WMC. Our criteria to classify similar use cases and to classify shared classes between use cases is shown in Table III, where \( X \) is the value of Jaro-Winkler metric and \( Y \) is the number of lines of code (LOC) of classes that are shared between use cases.

In order to assess the percentage of LOC sharing we will use the following formula:

\[
\frac{\text{number of Nonshared LOC}'s}{{\text{number of Shared LOC}'s}}
\]

**Metrics extraction**

In order to extract our chosen metrics we used the open source library Mono.Cecil\(^1\). Using this library, we calculated the values for WMC, CBO and LCOM.

### A. Research Questions

The main focus of our research is the analysis of the impact of use case names similarity in the sharing of classes between use cases and what kind of impact this sharing can have in the metrics WMC, CBO and LCOM.

The justificative for this study is that according analysis of use cases, we would predict what impact the similarity of use cases can have in software metrics that assess the design quality. We suggest that this analysis can help developers to prevent anomalies in software design, either in early stages or during maintenance.

**Research Question 1 (RQ1)** In similar uses cases, do the classes that implement them have higher percentage of shared lines of code (LOC) compared to dissimilar use cases? Our hypothesis is that if use cases \( u_1 \) and \( u_2 \) are considered similar, then they will have shared lines of code because we expect some reuse in the implementation of those use cases and that sharing is supposed to be higher in classes that implement similar use cases.

**Research Question 2 (RQ2)** In similar uses cases, do shared classes that implement them have lower cohesion compared to the other classes of the system? Our hypothesis is that if use cases \( u_1 \) and \( u_2 \) are considered similar, then classes that implement them have lower cohesion than the other classes of the system because although they work in similar problems they need to manage the variability of the use cases.

**Research Question 3 (RQ3)** In dissimilar uses cases, do shared classes that implement them have lower cohesion (higher LCOM) compared to the other classes of the system? Our hypothesis is that if use cases \( u_1 \) and \( u_2 \) are considered dissimilar and still have shared classes that implement them,

\(^1\)http://www.mono-project.com/Cecil
then those classes would have lower cohesion because they have responsibility to provide specialized services to dissimilar use cases.

Research Question 4 (RQ4) In dissimilar uses cases, do shared classes that implement them have higher complexity (WMC) compared to the other classes of the system? Our hypothesis is that if use cases \( u_1 \) and \( u_2 \) are considered dissimilar and still have shared classes that implement them, then those classes would have higher complexity because they have to cope with the variability of different use cases.

Research Question 5 (RQ5) In similar uses cases, do shared classes that implement them have lower coupling compared to the other classes of the system? Our hypothesis is that shared classes of similar uses cases perform similar activity and so use the same classes for that use cases, and this would require lower coupling.

Research Question 6 (RQ6). In dissimilar uses cases, do shared classes that implement them have higher coupling (CBO) compared to the other classes of the system? Our hypothesis is that shared classes of dissimilar uses cases perform different activities, and so, need to use different sources of classes and thus would have a high coupling with those classes.

Research Question 7 (RQ7). Is the cohesion of classes shared by dissimilar use cases lower than the cohesion of classes shared by similar use cases? Our hypothesis is that classes that implement dissimilar uses cases would perform distinct tasks, so they would have lower cohesion that classes in principle perform similar tasks.

IV. Results

In this section we present the results that will support the answers for the defined research questions.

The metrics collected from the subject system are shown in Table IV.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Mean</th>
<th>SD</th>
<th>Min</th>
<th>1Q</th>
<th>2Q</th>
<th>3Q</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOC</td>
<td>101.5</td>
<td>112.7</td>
<td>14</td>
<td>30</td>
<td>58</td>
<td>127.3</td>
<td>596</td>
</tr>
<tr>
<td>WMC</td>
<td>23.5</td>
<td>15.4</td>
<td>4</td>
<td>12.3</td>
<td>19</td>
<td>28.5</td>
<td>72</td>
</tr>
<tr>
<td>CBO</td>
<td>5.4</td>
<td>5.5</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>11.3</td>
<td>20</td>
</tr>
<tr>
<td>LCOM</td>
<td>16.1</td>
<td>13.6</td>
<td>0</td>
<td>7</td>
<td>11</td>
<td>25.5</td>
<td>64</td>
</tr>
</tbody>
</table>

In order to answer the research questions the Mann-Whitney U test was applied to analyze if the value of a metric in one data group is lower or higher than the other data group.

RQ1. According to the data shown in Figure 2, we applied the Spearman correlation test to analyze if the similarity of classes are correlated with the LOC sharing. The result is shown in Table V. Although we have a weak correlation (\( \rho = 0.1891 \)), that correlation is statistically significant at p-value 0.0086.

RQ2. Figure 3 shows the boxplot drawn for shared classes of similar use cases compared to other classes. Table VI shows that the cohesion of shared classes are lower than the cohesion of the other classes.

RQ3. Figure 4 shows the boxplot drawn for shared classes of dissimilar use cases compared to other classes. Table VII shows that the cohesion of shared classes are lower than the cohesion of the other classes.

RQ4. Figure 5 shows the boxplot drawn for shared classes of dissimilar use cases compared to other classes. Table VIII shows that the complexity measured by WMC of shared classes are higher than the complexity of the other classes.

RQ5. Figure 6 shows the boxplot drawn for shared classes of similar use cases compared to other classes. Table IX shows that there is no significant difference between the coupling measured by CBO of shared classes and the coupling of the other classes.

RQ6. Figure 7 shows the boxplot drawn for shared classes of dissimilar use cases compared to other classes. Table X shows that there is no significant difference between the coupling measured by CBO of shared classes and the coupling of the other classes.
TABLE VII. MANN-WHITNEY TEST FOR RQ3 - LCOM - DISSIMILAR

<table>
<thead>
<tr>
<th>Feature</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>P value</td>
<td>0.0317</td>
</tr>
<tr>
<td>One or two-tailed P value?</td>
<td>one-tailed</td>
</tr>
<tr>
<td>Are medians signif. different? (P &lt; 0.05)</td>
<td>Yes</td>
</tr>
<tr>
<td>Mann-Whitney U</td>
<td>64.50</td>
</tr>
</tbody>
</table>

So, we can see that the coupling of shared classes between use cases do not depend on the use case similarity.

**RQ7.** Figure 8 shows the boxplot drawn for shared classes of similar use cases compared to shared classes of dissimilar use cases. Table XI shows that the there is no significant difference between the cohesion in those classes.

The final result was that we could answer positively RQ1, R2, RQ3 and RQ4. However, contrary to our hypotheses, RQ5, RQ6 and RQ7 were answered negatively.

V. DISCUSSION

Our proposed automated analysis for use case similarity had a precision of 90% at 60% of recall. This result support our hypothesis that the identification of similar use case can be automated. The major benefit of this automated approach is higher productivity during this process. This is essential to engage developers in the adoption of the approach.

Another interesting finding was that the higher is the similarity of use cases, the higher is the LOC sharing between the classes that implement those use cases. This LOC sharing is not necessarily related to code clones, but related to the number of classes that are shared among the use cases.

Concerning the relation of shared classes and design metrics, some interesting findings could be presented.

We have observed that independently of the use case similarity, classes that are shared between different use cases tend to have lower cohesion. In this case, we need to have special attention to these classes and analyze the possibility of refactoring those classes. In fact, we can even suggest that during the development phase, developers pay more attention on classes being shared between different use cases.

We also have observed that the complexity measured with WMC of classes shared between dissimilar use cases tend to have higher complexity than classes shared between similar use cases. Just as in the case of cohesion, developers should pay attention on those classes and eventually try to find some pattern of refactoring that could be applied to those classes to enhance their overall WMC.

Concerning coupling of classes, we could not observe an interesting pattern that could call the attention of developers.
classes.html

Fig. 8. Boxplot for RQ7 results

TABLE XI. MANN-WHITNEY TEST FOR RQ7

<table>
<thead>
<tr>
<th>Feature</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>P value</td>
<td>0.5000</td>
</tr>
<tr>
<td>One or tow-tailed P value?</td>
<td>One-tailed</td>
</tr>
<tr>
<td>Are medians signif. different? (P &lt; 0.05)</td>
<td>No</td>
</tr>
<tr>
<td>Mann-Whitney U</td>
<td>53.50</td>
</tr>
</tbody>
</table>

VI. THREATS TO VALIDITY

The subject system used in this study have some features that could have some influence on the results. The use of POJO\(^2\) classes, i.e., classes that have only attributes and getters and setters, and the use of domain-driven development [7] may have influence on the coupling metric (CBO). Other studies with different domains and architecture would strength the value of our results.

Another threat was the use of developers to map use cases to classes and to evaluate the similarity of use cases. Because this is a manual process, mistakes could have occurred. We tried to mitigate this threat replicating the work of the developers and getting the consensus from them. Another mitigation criterion was the selection of experts that had participated in the implementation and documentation of the system, so their knowledge on the system would produce less error-prone results.

A threat to the external validity is the representativeness of the chosen subject system. This is a system that do not represent the large universe of software. So, our results are more likely to be valid with typical information management systems.

VII. CONCLUSION AND FUTURE WORK

In this paper, we have studied the impact that the similarity of use cases can have in LOC sharing of classes that implement them, and in metrics for coupling, cohesion and complexity. We found that there is a significant relation, although not strong, between use case similarity and LOC sharing. Interestingly, we also have found that classes that are shared between different use cases tend to have lower cohesion. Moreover, classes that are shared between similar use cases tend to have higher method complexity in classes. We found no relation between use case similarity and coupling. These findings can guide the developer, either in early development phases or in maintenance activities, to produce designs with higher cohesion and lower method complexity in classes.

We could extend this work in several ways, such as, reproducing the study with other subject systems and investigating the relation between use case similarity and code clones.

REFERENCES
