Concern-Based Cohesion as Change Proneness Indicator: An Initial Empirical Study

Bruno C. da Silva  
Computer Science Department  
Federal University of Bahia (UFBA)  
Salvador, Bahia, Brazil  
brunocs@dcc.ufba.br

Cláudio Sant’Anna  
Computer Science Department  
Federal University of Bahia (UFBA)  
Salvador, Bahia, Brazil  
santanna@dcc.ufba.br

Christina Chavez  
Computer Science Department  
Federal University of Bahia (UFBA)  
Salvador, Bahia, Brazil  
flach@dcc.ufba.br

ABSTRACT

Structure-based cohesion metrics, such as the well-known Chidamber and Kemerer’s Lack of Cohesion in Methods (LCOM), fail to capture the semantic notion of a software component’s cohesion. Some researchers claim that it is one of the reasons they are not good indicators of change proneness. The Lack of Concern-based Cohesion metric (LCC) is an alternative cohesion metric which is centered on counting the number of concerns a component implements. A concern is any important concept, feature, property or area of interest of a system that we want to treat in a modular way. In this way, LCC focus on what really matters for assessing a component’s cohesion - the amount of responsibilities placed on them. Our aim in this paper is to present an initial investigation about the applicability of this concern-based cohesion metric as a change proneness indicator. We also checked if this metric has a correlation with efferent coupling. An initial empirical assessment work was done with two small to medium-sized systems. Our results indicated a moderate to strong correlation between LCC and change proneness, and also a strong correlation between LCC and efferent coupling.

Categories and Subject Descriptors

D.2.8 [Software Engineering]: Product metrics; G.3 [Probability and Statistics]: Experimental design.

General Terms

Measurement, Design, Experimentation.

Keywords

Software metrics, cohesion, empirical software engineering.

1. INTRODUCTION

Cohesion is considered as an important internal quality attribute of software. A software module is said to be cohesive if it represents an abstraction of a single concept or feature of the problem domain. The fewer responsibilities a module has, the higher cohesive it is. Cohesion measurement has been claimed to be useful for assessing different aspects of software design, including prediction of module change proneness. Several metrics have been proposed so far with the purpose to quantify at what extent a module is cohesive or lacks cohesion. Considering object-oriented systems, most of the cohesion metrics are inspired from the well-known Lack of Cohesion in Methods metric (LCOM), proposed by Chidamber and Kemerer (C&K) [1].

Similarly to LCOM, most of the existing cohesion metrics only rely on the structural aspects of a class [2]. In order to quantify cohesion, they use syntactical information, such as pairs of methods that access the same attributes. As a consequence, they do not capture well the degree to which a module is cohesive or not in terms of the amount of responsibilities it has. This is one of the reasons why structure-based cohesion metrics are not good indicators of change proneness [3].

Few metrics have been proposed based on semantic information, such as: conceptual cohesion [2], which is based on the analysis of information embedded in the source code, such as comments and identifiers; and semantically-based cohesion [4], which is calculated using knowledge-based systems, program understanding and natural language processing techniques. However, studies on the correlation of these metrics with change proneness have not been carried out yet.

One of these new cohesion metrics is Lack of Concern-based Cohesion (LCC) [6]. It is part of a growing body of relevant work focusing on concern-driven measurement [5, 6]. Concern-driven metrics are centered on the abstraction of concern. A concern is any important property or area of interest of a system that we want to treat in a modular way [9]. Business rules, distribution, persistence, security and caching are examples of concerns found in many software systems.

The LCC metric is applied per component, such as a class. It captures whether a component is likely to be cohesive or not in terms of the number of concerns it implements. If a class contributes to the implementation of several concerns, it lacks cohesion. Regardless the syntactic structure of a class, this metric simply looks into what really matters for the implementation of the components - the amount of responsibilities (concerns) placed on them. LCC has been applied in a number of empirical studies with different goals [6, 7, 8, 10].

The reasoning behind LCC is that a component that encompasses a large number of concerns is change prone. This is because it may suffer from modifications derived from change requests related to any of the concerns implemented by it. Furthermore, one would expect to find a correlation between efferent coupling and LCC, because if a class has many concerns it is likely to
depend on other classes as it participates more on the overall system’s concerns.

However, those are merely informal assumptions which have not been validated yet. Therefore, the main goal of this paper is to present an initial empirical assessment of the correlation between LCC and change proneness. This paper also aims at studying the correlation between LCC and efferent coupling, in particular the Coupling Between Objects metric (CBO) [1]. We applied LCC, LCOM, and CBO in the first version of two systems, then we analyzed their further change history, and obtained the following results: (i) a moderate to strong correlation between LCC and change proneness in one system, and a similar correlation strength between LCOM and change proneness in the second system; (ii) also we obtained a strong correlation, as expected, between LCC and CBO in both systems, and a moderate correlation between LCOM and CBO only in one system; (iii) finally, we also highlighted and discussed important particular situations that may have negatively affected those correlation tests.

The remainder of this paper is organized as follows: Section 2 explains the LCC metric; while Section 3 describes a motivating example regarding the LCC metric and its association with change proneness; in Section 4, we present the empirical study including explanations of the target applications, applied metrics and quantitative results; Section 5 is focused on a qualitative discussion about interesting observed situations; Section 6 points out the study constraints; and Section 7 presents the final remarks and next steps.

2. LACK OF CONCERN-BASED COHESION METRIC (LCC)

 Concern-driven metrics [5, 6] capture design properties associated with the realization of concerns in software artifacts. Concern-driven metrics rely on a concern-to-code mapping. The mapping consists of assigning a concern to the corresponding source code elements (e.g. method and classes) that realize it. Therefore, before computing concern-driven metrics, it is necessary to identify the code fragments responsible for implementing each concern in the system.

Sant’Anna [6] defined LCC in order to quantify cohesion of a given component in terms of the quantity of concerns addressed by it. A component can be a class, an interface, or whatever represents a module as a unit of implementation. So, the results of this metric are obtained per component. Thus, it counts the number of concerns mapped to each component. Although it was firstly defined for architectural models, it can also be used at the implementation level with no constraints.

To express the metric unambiguously and facilitate the replication of our empirical study, we present the formal definition of LCC based on set theory, as it is presented in [7]. First, we present the terminology used on the formal definition. Let S be a system, the classes and interfaces of S are called as components and denoted by \(C(S)\). Each component \(c\) consists of a set of attributes, denoted as \(A(c)\), and a set of operations, represented as \(O(c)\). The set of all attributes and all operations in system \(S\) are represented as \(A(S)\) and \(O(S)\), respectively. For each \(c \in C(S)\), the set of concerns assigned to \(c\) is denoted as \(Con(c)\). Let \(o \in O(c)\) be an operation of \(c\), the set of concerns assigned to \(o\) is denoted as \(Con(o)\). Let \(a \in A(c)\) be an attribute of \(c\), the set of concerns assigned to \(a\) is denoted as \(Con(a)\). For each concern \(con\) realized on the design of system \(S\), the set of components, operations and attributes to which \(con\) is assigned is, respectively, denoted as:

\[C(con) = \{c \mid c \in C(S) \land con \in Con(c)\}, O(con) = \{o \mid o \in O(S) \land con \in Con(o)\}, \text{and } A(con) = \{a \mid a \in A(S) \land con \in Con(a)\}\].

The LCC metrics can now be defined as follows:

\[LCC(c) = \bigcup_{o \in O(c)} Con(o) \bigcup_{a \in A(c)} Con(a)\]

3. MOTIVATING EXAMPLE

Here we present a concrete example that motivates the assumption that the LCC metric can be used as a change proneness indicator. We took a class (called BaseController) from the MobileMedia system [11]. MobileMedia is a system for managing photos, music and videos in mobile devices. It is one of the target systems of our empirical study (see Section 4.1). The BaseController class contributes to the implementation of five concerns in the first version (V1) of the system. So \(LCC = 5\) for this class. This can be considered as a high value as it includes all the concerns mapped at this version of the system. Those concerns are: Controller, Exception Handling, Label, Persistence and Photo.

Taken the V1 as the start point, and the evolution up to the sixth version (V6), the BaseController class was modified many times, totalizing 32 operation-related changes and 6 attribute-related changes. In V2, the MobileMedia system was modified by two main reasons: (i) by the inclusion of a new feature for sorting photos manipulated by the system, and (ii) by the changes necessary for allowing the edition of the photos’ label. As BaseController class contributes to the implementation of the Photo and Label concerns, it was affected by such modifications.

In V3, the system incorporated the implementation of another new feature, which allows users to specify and view their favorite photos. Again as BaseController contributes to the implementation of the Photo concern, it was changed due to this modification on the system.

In V4, the Controller concern was restructured. This concern refers to the control of the interaction between the view classes and the model classes of the system. Before V4, this control was centered in the BaseController class. In V4, it was split into four other classes: AbstractController, AlbumController, PhotoListController and PhotoViewController. Because of this BaseController suffered a lot of changes: 26 operation-related changes and 6 attribute-related changes. As a result, these modifications left BaseController in V4 with only to 2 concerns: Controller and Photo.

In V5, BaseController was not modified, but, finally, in V6 suffer the last changes. Two new features were introduced to the systems to allow the user to also manipulate music and video. Those modifications directly affected the Photo concern, as Photo was turned into an alternative feature of the product line. Again, this involved changes in the BaseController class as it encompasses the Photo concern.

In the Mobile Media evolution history, we can note that the BaseController class, which started in V1 with a high LCC value, suffered a relatively high number of changes on the course of six
versions. Therefore, we could say that the **BaseController** reveal to be a change-prone class in the first version of the system.

### 4. EMPIRICAL STUDY

This section presents the empirical study we have conducted to analyze the correlation between the LCC metric and software components’ change proneness. This study also served to observe the correlation between LCC and coupling (CBO metric). We also included the LCOM metric in the study in order to compare its correlation results with LCC ones. In the following sections, we present the study settings including a description of the analyzed systems (Section 4.1) and how we performed the measurements (Section 4.2). Subsection 4.3 focuses on the performed correlation tests and their results.

#### 4.1 Target Applications

The study involved two applications, named Mobile Media (MM) and Health Watcher (HW). They are described as follows.

**Mobile Media**

MobileMedia is a Software Product Line (SPL) for applications with about 4 KLOC that manipulate photo, music, and video on mobile devices, such as mobile phones. It was developed based on a previous SPL called MobilePhoto [13]. In order to implement MobileMedia, the developers extended the core implementation of MobilePhoto including new mandatory, optional, and alternative features. The alternative features are just the types of media supported: photo, music, and/or video. Examples of core features are: create/delete media, label media, and view/play media. In addition, some optional features are: transfer photo via SMS, count and sort media, copy media and set favorite photos.

The MM concerns we considered in our study were the same ones used in a previous study [11]. The application of LCC was based on the concern-to-code mapping undertaking on the same previous study [11]. Table 1 summarizes the evolution history of the MM system. It briefly describes the changes made in each version considered in this study. The change scenarios comprise different types of changes including mandatory, optional, and alternative features of the MM product line.

<table>
<thead>
<tr>
<th>V#</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>New feature added to count the number of times a photo has been viewed and sorting photos by highest viewing frequency. New feature added to edit the photo’s label.</td>
</tr>
<tr>
<td>3</td>
<td>New feature added to allow users to specify and view their favorite photos.</td>
</tr>
<tr>
<td>4</td>
<td>New feature added to allow users to keep multiple copies of photos.</td>
</tr>
<tr>
<td>5</td>
<td>New feature added to send photo to other users by SMS.</td>
</tr>
<tr>
<td>6</td>
<td>New feature added to store, play, and organize music. The management of photo (e.g. create, delete and label) was turned into an alternative feature. All extended functionalities (e.g. sorting, favorites and SMS transfer) were also provided.</td>
</tr>
</tbody>
</table>

Table 1. MobileMedia evolution history

We selected the MobileMedia application as one of our objects of study for several reasons. First, it is a non-trivial system implemented in Java with multiple releases available, each of them introducing realistic, heterogeneous change scenarios. Also, all requirements, architecture, and implementation artifacts are available for all releases, which allow possible extensions of our study using those artifacts. Second, the MM design is particularly rich in several kinds of concerns, including mandatory, optional, and alternative features as well as nonfunctional requirements. Third, the MM design were developed with modularity and changeability principles as main driving design criteria and they were extensively discussed in a controlled manner [11].

**Health Watcher**

The second target application is a medium-sized Web-based information system, called Health Watcher [14]. The Health Watcher system allows a citizen to register complaints to the public health system. Complaints are registered, updated, and queried through a Web client. The system is structured in layers with the goal of decoupling different parts of the system and making these parts easy to change independently. Concurrency, persistence, distribution and exception handling were the HW concerns we considered in our study. These were the same concerns took into account by [14] in their study. In order to compute LCC in our study, we also considered the same concern-to-code mapping used by [14].

The selected changes of the Health Watcher system vary in terms of the types of modifications, as summarizes Table 2. Some of them add new functionality, some improve or replace functionality, and others improve the system structure for better reuse or modularity. These changes originate from many sources. For instance, the original developers of HW implemented changes to meet new stakeholders’ requests (that are actually necessary) or just to improve the system’s structure (that are not compulsory). There are also changes created by the students and researchers involved in previous studies [14], where certain extensions and improvements were implemented. Before the changes have been applied, the original developers of Health Watcher were consulted to confirm whether these changes were valid.

<table>
<thead>
<tr>
<th>V#</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>Factor out multiple Servlets to improve extensibility</td>
</tr>
<tr>
<td>3</td>
<td>Ensure the complaint state cannot be updated once closed to protect complaints from multiple updates</td>
</tr>
<tr>
<td>4</td>
<td>Encapsulate update operations to improve maintainability using common software engineering practices</td>
</tr>
<tr>
<td>5</td>
<td>Improve the encapsulation of the distribution concern for better reuse and customization</td>
</tr>
<tr>
<td>6</td>
<td>Generalize the persistence mechanism to improve reuse and extensibility</td>
</tr>
<tr>
<td>7</td>
<td>Remove dependencies on Servlet response and request objects to ease the process of adding new GUIs</td>
</tr>
<tr>
<td>8</td>
<td>Generalize distribution mechanism to improve reuse and extensibility</td>
</tr>
<tr>
<td>9</td>
<td>New functionality added to support querying of more data types</td>
</tr>
</tbody>
</table>

Health Watcher was selected because it met a number of relevant criteria for our study. First, it is part of a real-world health care system used by the city of Recife in Brazil. Also, it has around 7000 lines of code. The HW system complements MobileMedia (our first application) with different kinds of concerns present in typical web-based information systems. HW also involves a number of recurring concerns and technologies common in day-
to-day software development, such as GUI, Persistence, Concurrency, RMI, Servlets, and JDBC. Also, the HW design and implementation choices have been extensively discussed [14] and evolved in a controlled manner. As MobileMedia, the Health Watcher system was also developed with modularity and changeability principles in mind. Finally, the first HW release of the Java implementation was deployed in March 2001, since then a number of incremental and perfective changes have been addressed in posterior Health Watcher releases.

4.2 Measurement Procedures

This study involved the application of four metrics: (i) Lack of Concern-based Cohesion (LCC), which was described in Section 2; (ii) Coupling Between Objects (CBO), defined by Chidamber and Kemerer (1994); (iii) Lack of Cohesion in Methods (LCOM), also defined by Chidamber and Kemerer [1]; and Number of Changes, to which we gave the NCh acronym.

We applied the LCC, CBO and LCOM for the first version of the two systems. Then we applied the NCh metric, which counts, for each component in the first version, the number of subsequent versions on which that component underwent changes. For instance, if a component was changed in versions 2, 5 and 9 of HW, then NCh for this component is three. The goal was to evaluate whether the value of LCC and LCOM of the first version could indicate which components would change more in the course of the evolution of the systems. Also, we aimed at evaluating whether the values of LCC and LCOM in the first version correlates somehow with the values of CBO in the same version.

We computed the LCC metric based on the concern-to-code mapping files previously prepared during the studies presented in [11] and [14]. During these studies, a group of six researchers, grouped in three pairs, manually identified which fragments of code contributes to which concerns. The obtained mapping files list for each concern the components that contribute to its implementation. The LCOM and CBO was also computed during the studies presented in [11] and [14] with the support of a tool. In order to compute the NCh metric, we used a diff tool to support us on identifying the classes that was changed from one version to the next one. We ignored changes related to source code comments or code layout rearrangement.

4.3 Statistical Tests

Before applying the statistical correlation tests, it is necessary to know about the normality of the data to be tested, which, in our case, were the data gathered from the application of the metrics in the source code of the HW and MM systems. Hence, to check if the measurements revealed normal distribution we plotted histograms corresponding to each measurement for HW and MM. We analyzed visually the histograms trying to know more about the distributions the measurements shape. Figure 1, charts (a) and (b), shows the histograms of LCC in MM and HW as an example, which we found most likely to be a normal distribution by analyzing them visually.

Figure 1, charts (c) and (d), illustrates histograms of NCh in MM and LCOM in HW as examples of non-normal distributions. After a visual analysis looking into the graphs, we used the SPSS tool [15] to apply the Kolmogorov-Smirnov and Shapiro-Wilk normality tests in all the measurements from HW and MM, respectively, to check if they are normal distributions. The Kolmogorov-Smirnov normality test is indicated when the sample has more than 30 points. It was the case of HW which has 88 components for each applied metric. On the other hand, MM has only 24 components for each applied metric. So we applied the Shapiro-Wilk normality test. As a result of those normality tests, we encountered only the LCC in MM as a normal distribution. Even LCC in HW system, which looks as a normal distribution from the chart (Figure 1b), revealed to be a non-normal distribution by the statistical test.

4.4 Results from the Correlation Tests

The selection of the correlation tests depends on the results of the normality analysis. As explained in the previous subsection, most of the measurements for both systems shape a non-normal distribution. Actually, only the LCC measurement in MM is a normal distribution. Thus, as we had non-normal distributions in all the correlation comparisons, we apply the Spearman correlation method [15] (using SPSS). Tables 3 and 4 summarize the results for MM and HW, respectively. They show the correlation matrix obtained from each system. The correlation matrix is symmetric and each cell contains the correlation coefficient value (top) and its significance level (bottom). All the correlations with significance level less than .05 are in bold, as we can only make conclusions over those numbers. It is worth to recall that the correlation coefficient value ranges from -1 (a perfect negative correlation) to 1 (a perfect positive correlation). A zero correlation coefficient between two variables means that such variables are not associated at all. In order to give a qualitative label for the obtained correlation numbers we follow [17], which suggests that a coefficient: from .00 to .30 is a weak correlation; from .30 to .50 is a moderate correlation; and greater than .50 is a strong correlation. This classification is also used in other works concerning measurements correlation [5] [18].
## 5. DISCUSSION

As expected we found a strong correlation between LCC and NCh, according to Cohen’s ranges (>0.5 means strong) [17]. However, the correlation tests actually showed a coefficient (.518) very close to the threshold limiting high and moderate. That’s why we claimed in the introduction that we found a moderate to strong correlation. In this section, we present and discuss in details some examples that contributes to the correlation not be so strong. These are examples of classes with high LCC value and low number of changes, and classes with low LCC values and high number of changes. Additionally, we comment the issue of the negative correlation between LCC and LCOM.

### High LCC and No Changes

We found some classes in both systems that have a high LCC value but did not suffered any change or suffered very few changes. For instance, the MM system has some classes for handling exception when some data are inserted to or requested from the data repository. In the first version of MM (V1), there are six of these classes. For instance, the `ImageNotFoundException` class represents an exception that may occur when a requested photo is not found in the repository. Another example is the `InvalidPhotoAlbumNameException` class which represents an exception that occurs when the user tries to include an album of photos with an invalid name. Each of these classes encompasses three concerns in the first version of the system: exception handling, persistence and photo. Therefore, they have LCC = 3. We consider this as a high value as the first version of MM has a total of only five concerns (besides the aforementioned ones, MM also has the controller and label concerns).

Although having a high LCC value, the mentioned classes suffered no changes along the versions because of two reasons. First, the change scenarios (Table 1) did not exercise two of the three concerns: exception handling and persistence. There was no change directly related to these two concerns. The second reason is because the six exception classes should have been changed in version 6 but the programmers forgot or decided not to do that. The changes in V6 exercised the `Photo` concern: a new feature – Music – were added and the mandatory feature `Photo` was modified to be turned into an alternative feature. At this moment, the six exception classes should have been at least renamed in order to represent not only exceptions related to Photo but also related to Music. For instance, the `ImageNotFoundException` class should have been renamed to `MediaNotFoundException`. However, the programmers kept it (and the other exception classes) with the original name, even though they were now used for Photo and Music.

### Low LCC and Several Changes

Another situation that contributed negatively to the correlation between LCC and NCh are classes with low LCC value which were modified in several versions. The `PhotoListScreen` class in the MM system is an example of such case. This class is

### Table 3. Correlation matrix from MM measurements

<table>
<thead>
<tr>
<th></th>
<th>NCh</th>
<th>LCOM</th>
<th>CBO</th>
<th>LCC</th>
</tr>
</thead>
<tbody>
<tr>
<td>NCh</td>
<td>1</td>
<td>.564*</td>
<td>.618**</td>
<td>.295</td>
</tr>
<tr>
<td>Correlation Coefficient (Sig. 2-tailed)</td>
<td>.086</td>
<td>.584</td>
<td>.000</td>
<td></td>
</tr>
<tr>
<td>LCOM</td>
<td>.646**</td>
<td>.518*</td>
<td>.000</td>
<td>.000</td>
</tr>
<tr>
<td>Correlation Coefficient (Sig. 2-tailed)</td>
<td>.000</td>
<td>.087</td>
<td>.000</td>
<td></td>
</tr>
<tr>
<td>CBO</td>
<td>.518*</td>
<td>.307</td>
<td>.694**</td>
<td>1</td>
</tr>
<tr>
<td>Correlation Coefficient (Sig. 2-tailed)</td>
<td>.000</td>
<td>.045</td>
<td>.000</td>
<td></td>
</tr>
</tbody>
</table>

**. Correlation is significant at the 0.01 level (2-tailed).
* Correlation is significant at the 0.05 level (2-tailed).

### Table 4. Correlation matrix from HW measurements

<table>
<thead>
<tr>
<th></th>
<th>NCh</th>
<th>LCOM</th>
<th>CBO</th>
<th>LCC</th>
</tr>
</thead>
<tbody>
<tr>
<td>NCh</td>
<td>1</td>
<td>.086</td>
<td>.646**</td>
<td>.518*</td>
</tr>
<tr>
<td>Correlation Coefficient (Sig. 2-tailed)</td>
<td>.584</td>
<td>.087</td>
<td>.000</td>
<td></td>
</tr>
<tr>
<td>LCOM</td>
<td>.000</td>
<td>.000</td>
<td>.000</td>
<td></td>
</tr>
<tr>
<td>Correlation Coefficient (Sig. 2-tailed)</td>
<td>.000</td>
<td>.000</td>
<td>.000</td>
<td></td>
</tr>
<tr>
<td>CBO</td>
<td>.518*</td>
<td>.307</td>
<td>.694**</td>
<td>1</td>
</tr>
<tr>
<td>Correlation Coefficient (Sig. 2-tailed)</td>
<td>.000</td>
<td>.045</td>
<td>.000</td>
<td></td>
</tr>
</tbody>
</table>

**. Correlation is significant at the 0.01 level (2-tailed).
* Correlation is significant at the 0.05 level (2-tailed).

### LCC vs. NCh and LCOM vs. NCh

From the results obtained from MM, we are not able to say whether or not LCC has correlation with NCh. This occurs because the significance level obtained was .162 (higher than .05). In other words, we cannot conclude anything about the use of such a metric as change proneness indicator. However, in the HW case, the test obtained a satisfactory significance level with a correlation coefficient .518, which can be considered as a strong correlation.

The inverse occurred with the correlation between LCOM and NCh. The MM scenario produced a satisfactory significance level with a strong correlation (coefficient of .564), while in the HW case the test did not produce a significance level sufficient to make conclusions.

### LCC vs. CBO and LCOM vs. CBO

The results for LCC vs. CBO followed our initial expectations for both systems. In the MM case, we found a strong correlation (coefficient .504), and for HW we found an even stronger correlation with coefficient .694.

Regarding the relationship between LCOM and CBO, the correlation tests found a moderate correlation (coefficient .488) between LCOM and CBO in the MM system. However, in the HW case the test did not found a significance value enough to make conclusions about this correlation.
responsible for listing the photos in the mobile devices screen. It is also responsible for showing in the screen the menu items related to the actions the users can perform with the photos, for instance, delete a photo. The only concern mapped to this class in the first version of the system was the Photo concern. Therefore, LCC = 1 for it. However, this class was modified in three versions (V2, V3 and V6), totaling six operation-related changes and eight attribute-related changes.

The changes suffered by PhotoListScreen in V6 were already expected because this version involved modifications related to the Photo concern: the Photo feature was turned into an alternative feature with the inclusion of the Music feature, as explained before. However, one may say that the changes suffered by PhotoListScreen in V2 and V3 were not expected, as they were not directly related to the Photo concern, the only concern in this class. In fact, V2 and V3 involved the inclusion of new features to the system. These new features represented two new concerns for the system: Sorting and Favorite. In V2, a feature for sorting the listed photos was added. In V3, the feature for specifying favorite photos was included. These changes affected the PhotoListScreen class because new menu items had to be showed in the screen in order to allow the user to access the new features.

Based on this example, we may conclude that LCC is not a good predictor of changes that involves the inclusion of new concerns. However, analyzing from a different perspective, we can say that the Screen concern should have been considered in the concern-to-code mapping process and it should have been assign to the PhotoListScreen class. Remember that this class is also responsible for displaying menu items on the screen. Therefore, it seems that the concern-to-code mapping is incomplete, which affected the measurement results. If the Screen concern had been considered and mapped in the MM system, the PhotoListScreen class would have a higher LCC value which, therefore, would be closer to its observed change prone behavior.

**Imprecise Concern-to-Code Mapping**

The used concern-to-code mapping strategy certainly affects concern-based measurements and the correlation between LCC and change proneness. We could clearly observe in MM and HW that there were some classes with code fragments not assigned to any of the considered concerns. Moreover, some classes had LCC = 0, which means that no concern was mapped to them. As a result, changes in these classes might be related to unmapped or “unknown” concerns, such as the aforementioned Screen concern. Surely this phenomenon causes confusion on the correlation between LCC metric and change proneness. In fact, mapping concern-to-code is a time-consuming, subjective and error-prone activity. Therefore, having a precise and complete mapping is not trivial and requires tool support. Also it is hard to obtain the same mapping twice, which hinders the repeatability of measurement results. Actually, researchers already know that this issue represents the main drawback of concern-based metrics and should be carefully studied. Studies on this have been recently carried out [12] [16]. However, this discussion is out of the scope of this paper.

**Negative Correlation between LCC and LCOM**

It is important to discuss the correlation between LCC and LCOM, although it is not the primary goal of the study. In the HW system we obtained a negative moderate correlation (with coefficient -0.307) between those measurements, while in the MM there was no significance level for this pair-wise comparison (see Tables 3 and 4). With these results in mind, we analyzed some HW measurements for LCC and LCOM individually. Then we made some observations which corroborate the correlation test results.

Firstly, it is important to notice that the LCOM metric is only applicable when classes have fields, as it counts how many pairs of methods access fields in common. Therefore, as we have several components in HW without fields (actually half of total), the domain of comparisons was halved. Thus, the correlation test made 44 pair-wise comparisons in HW; instead of 88. It also happened in the MM system, but in this case only 4 from the 24 components were eliminated from the comparison.

In addition, there are classes in the HW system with high LCC and low LCOM. For instance, LCC has value four for the HealthWatcherFacade class, which means that this class contributes to the implementation of the four concerns. However, although in the top LCC ranking, the HealthWatcherFacade class has LCOM = 0, which means that LCOM did not capture the fact that such class has several responsibilities. This occurs because most methods in this class propagate their call to calls to methods of HealthWatcherFacadeInit. For this, almost every method in HealthWatcherFacade accesses the field representing HealthWatcherFacadeInit. As a consequence, several semantically unrelated methods end up accessing a field in common, decreasing the value of LCOM.

On the other hand, we observed some classes with high LCOM values but zero LCC. For instance, the FoodComplaint class in HW has LCOM = 78 and LCC = 0. The discrepancy can partially be explained by the aforementioned problem of imprecise concern-to-code mapping, as there is not a Food or Complaint concern mapped to the HW. Therefore, a more complete concern-to-code mapping would assign to FoodComplaint those concerns, increasing the LCC value for it. On the other hand, the lack of cohesion of this class is not as high as the LCOM value suggests. In fact, it is a cohesive class. However, it encompasses several fields and several “setters” and “getters” methods, each of them accessing just one of the fields. This substantially increases the value of LCOM, but does not mean that the class is not cohesive.

**6. STUDY CONSTRAINTS**

This section discusses some constraints of the design and execution of our empirical study. The conclusions obtained here are restricted to the involved software systems as well as their change scenarios. In addition, the MM and HW systems, although real, are small to medium systems, which limits the obtained results. These are some of the reasons that we considered our study as initial. The main purpose of this initial study was to check if LCC was promising to be a change proneness indicator and deserved further investigation.

Another important point to highlight is the small number of versions considered in the change history – five versions for the MM system and eight versions for the HW system. A finer-grained version history would be better to generate a larger amount of data, especially for the NCh metric. However, it was not possible to obtain such finer-grained versions because the repository commits for the studied systems were done sparsely in time by the original developers.

Moreover, the small size of the systems (notably MobileMedia, which has only 22 classes in the first version) and the small
number of releases, together with the low variance of some measurement values, might have led to a non-significant level in some correlation comparisons.

Our study was also constrained by the quality of the concern-to-code mapping. In order to minimize this limitation, we decided to use mappings already used in previous studies [11, 14]. However, this revealed not to be a guarantee of having precise and complete mappings, as discussed in Section 5.

7. FINAL REMARKS AND NEXT STEPS

This work represents a first stepping stone towards the analysis of: (i) the correlation between the Lack of Concern-based cohesion metric and change proneness, and (ii) its applicability as a change proneness indicator. We carried out an empirical evaluation with the support of statistical tests and two systems as objects of study. We found a moderate to strong correlation between concern-based cohesion and change proneness. We also highlighted and discussed particular situations that may negatively affect this correlation. We considered that our findings showed that LCC is worth to be further investigated.

As it was a preliminary work, our studied was limited by several constraints as discussed in Section 6. Therefore, as future work we intend to replicate this study in a more controlled way. Firstly, we plan to use a higher number of larger systems with a higher number of releases. This can be achieved by taking the advantages of mining repository tools.

Secondly, with a higher number of releases at hand, it should be also interesting to take intermediate releases to see how the cohesion measurements change along the releases. It will certainly enrich the discussion about how cohesion is related to software changeability.

Moreover, we are studying semi-automated techniques for performing the concern-to-code mapping. We expect that these techniques will support us to: (i) produce more complete and precise mappings, and (ii) perform repeatable concern-driven measurements of larger systems in acceptable time.

8. ACKNOWLEDGMENTS

This work is partially supported by the National Institute of Science and Technology for Software Engineering (INES), funded by CNPq (grant 573964/2008-4). Claudio is also supported by CNPq under grant 480374/2009-0.

9. REFERENCES