Exploring the Combination of Software Visualization and Data Clustering in the Software Architecture Recovery Process

Renato Paiva, Genaína N. Rodrigues, Marcelo Ladeira, Rodrigo Bonifacio
Computer Science Department
University of Brasília
Brasilia, Brazil
{renatoedesio, mladeira, rbonifacioatcic}@unb.br,
genaina@cic.unb.br

ABSTRACT

Modernizing a legacy system is a costly process that requires deep understanding of the system architecture and its components. Without an understanding of the software architecture that will be rewritten, the entire process of reengineering can fail. When there is absence of architectural documents, it is important to have a recovery process of architecture that allows the complete understanding of the software. Such process involves mapping of source code entities in high-level models. Previous work using visualization and clustering techniques has been proposed and extensively used. However, their accuracy to reconstruct the architecture alone has shown to be not satisfactory enough. Thus, this work proposes to explore if an approach where visualization and clustering applied together can provide a higher accuracy on the software architecture recovery process. An experimental study was conducted to empirically evaluate our investigation. The results indicated a statistically significant increase in the accuracy of the models produced.

CCS Concepts

•**Computer systems organization** → **Embedded systems;** *Redundancy;* Robotics; •**Networks** → Network reliability;

Keywords

Software Architecture Recovery; Software Visualization; Data Clustering

1. INTRODUCTION

In order to evolve a software, it is paramount to understand its architecture. However, when there is a lack of architecture information, some reverse engineering practices must take place. The ability to mine relevant information from execution traces tend to quickly become large and unmanageable. Thus, a relevant challenge is to filter them and extract information relevant for the particular understanding of the task being performed [1]. Additionally, in most cases, understanding the software structure is a challenging task due to the lack of architecture specification documents, particularly w.r.t legacy systems. If there is no such specification available or if it's out-of-date, a software architecture recovery process must take place.

The required effort to manually recover a system architecture is proportional to the the size and complexity of its source code. This is particular more costly for large software systems, prone to every sort of errors [22]. As a result, the study of methods and tools to extract the software systems architecture is fundamental to guarantee a faster and more accurate process. By these means, costs and risks in the process of architecture recovery should be diminished.

In the context of architecture recovery techniques, a widely used technique is software architecture visualization [3, 4, 8, 17]. The basis of the software visualization underlies on the creation of an image of the system via visual objects [17]. Using such abstraction, it is possible to obtain a new view of the software, which allows exploring different concepts and clearer understanding of the software system structure.

Wettel *et al.* [20] carried out a controlled experiment to investigate the efficacy and effectiveness of the CodeCity software visualization tool in the process of understanding a software system structure. Results pointed out that, using such tool, it was possible to obtain an accuracy of 24.26% and a reduction of 12.01% in the execution time of a certain number of tasks, when compared to an understanding process carried out via a manual inspection of the source code.

Despite the improvement on the results obtained through software visualization techniques, most of the times, the efficiency of the methods or tools rely on the ability of the analyst in charge of the software architecture recovery. As a result, the accuracy of the results is bound by subjective criteria most of the times. In order to fill in this gap, various research work has been devoted to automate the architecture

SAC'16, April 4-8, 2016, Pisa, Italy

http://dx.doi.org/xx.xxxx/xxxxxxx.xxxxxxx

recovery process.

Another technique to recover a software system architecture consists on clustering methodologies [5,12,15,21]. Such technique is used in various fields of research to describe methods to cluster unclassified data. In the context of architecture recovery, clustering aims at grouping software artifacts in significant modules, leading towards an understanding of the software system structure in a higher abstraction level [15].

However, to decompose a software system in higher-level structures in a coherent and logic coupling, is an inherently difficult task. Due to the usual sparse set of data in a source code and the various computational factors involved in a software system, many solutions are only suboptimal and do not represent the real software architecture [12]. On the other hand, results of such technique are still helpful as they provide as a result a view of the software system decomposition, instead of reconstructing it from scratch. Further refinements are then required in order to have an accurate architecture recovery [18].

Given the inaccuracy inherent to architecture recovery techniques, one question comes out: Is it possible to obtain a more accurate software architecture recovery process? We propose to investigate this question by merging both visualization and clustering techniques into one architecture recovery process. We postulate that clustering and visualization techniques are complementary in the sense that they provide different levels of software architecture abstraction. Our investigation is performed on a controlled experiment in a industrial environment, where participants are software analysts and developers working on software systems developed in Java and Visual Basic programming languages. Our results show a statistically significant improvement in the accuracy of the models produced from our proposal compared to the techniques used alone.

The remaining sections of the paper are structured as follows: Section 2 presents the background concepts regarding software architecture visualization and clustering techniques. Section 3 highlights the inaccuracies of such techniques and the principles that drive our investigation. Section 4 is the main section where we present and evaluate the experimental study we conducted with the software development group in the Data Processing Center at University of Brasilia. In Section 5 we present the related work and finally, in Section 6 we conclude our work and present the future directions we plan to pursue.

2. RELATED WORK

The maintenance and evolution of systems is expensive and involves a high risk, which is due to the difficulty of understanding the system architecture. To solve this problem, several approaches for the recovery of software architecture have been proposed. Ducasse et al. in [2] present a taxonomy of approaches to architecture recovery, detailing information necessary for recovery, such as: what are the stakeholders' goals, how does the general reconstruction proceed, what are the available sources of information, based on this, which techniques can we apply, and, finally, what kind of knowledge does the process provide.

Software visualization provides complementary perspectives on the analyzed system. Through the use of software visualization, it is possible to explore the software architecture through representations based on diagrams "box-line", interconnected graphs, dependency matrices etc. These artifacts represent interactions between the modules in the system and can combine different metrics to increase the level of information of each diagram [9].

Lanza in [8] presents a tool that focuses on the software visualization. This tool supports reverse engineering through combinations of metrics and software visualization techniques. The author makes use of the concepts of polymetric views to facilitate the understanding of the software. In this type of view it is possible to have notions of how, when and what metric a specific class was represented. Figure **??** highlight the main screen of CodeCrawler tool.

The works of Sangal [14] and Tekinerdogan [16] are focused on representation of the source code in graphs and matrix structured dependencies, to create the system architecture of understanding of software. By analyzing such artifacts extracted through code static analysis, they show how to detect structural elements of the system and get an understanding of its architecture.

Regarding the use of data clustering for understanding software systems, the tool developed by Mancoridis *et al.* [10] has a notable highlight. Through that tool it is possible to automatically create breakdowns of the system structure in significant subsystems. The processed data can provide structural information to developers, facilitating the understanding of software components, their interfaces and interconnections. The tool provides a complete environment for the clustering process; implemented through a tool with a graphical interface and made available for free, with operating manual and sample code.

Another approach to the understanding of software systems by using data clustering techniques was presented by Tzerpos *et al.* [19]. In their study, it is presented a clustering algorithm that discovers clusters that follow patterns that are commonly observed in decompositions of large software systems that were prepared manually by their system architects. Also, the algorithm assign meaningful names for cluster generated.

3. RATIONALE ON THE COMBINATION BETWEEN VISUALIZATION AND DATA CLUSTERING

In many cases, the analyst, with some knowledge of a system, can perform an analysis of the results and create a concise final model. However, especially for complex systems, it is necessary to use strategies for interpretation of results. Such strategies involve the observation of repeated patterns and identification of architectural violations in the source code.

In general, automatic architecture reconstruction methods, such as clustering technique, has the advantage to produce different models for a single software system in a short period of time. Such models can be constructed differently by changing configuration settings on clustering algorithm used

Table 1: Occurrences of entities in the results.

Relation	Occurrence
{1,2,3}	100%
{1,2,3,4}	25%
{4,5}	50%
{4,5,6}	25%
{5}	25%
{6}	25%
{6,7,8,9}	50%
{7,8,9}	100%

during the process. Through the analysis of different models it is possible to identify patterns that recur frequently in the results.

However, when the idea is not clear of how the system structure is composed, the various models produced by clustering process can not help to understand complex software [11], since the results show a high level view of the architecture. In this sense, the use of software visualization technique permit an observation of different outcomes on a low level of abstraction. By means of interactive operations in the models produced by the visualization software, it is possible to decompose components of the system in more detailed representations. Thus, allowing the observation of concepts as part of the architecture in greater depth.

In this context, linking the models produced by the clustering process with the model produced by the software visualization process, it is possible to get different representations of the system to form a final model with greater precision. For example, take into account the results shown in Figure 1. Assuming that the process of obtaining architecture models recovered four different results, through the use of software visualization technique and clustering algorithms. Each model features 9 entities, namely: {1,2,3,4,5,6,7,8,9}. By performing a count of the grouped entities it is possible to obtain the ratio of how many times the entity was classified similarly to the other. The result of this count is observed in Table 1.

By analyzing the results, it is possible to see that the entities {1,2,3} can be classified into a single module, since they appear 100% of the times in the same relation; so do the entities {7,8,9}. The entities {4,5,6} may be classified as either a single module or each entity may be added to any other adjacent module, since there is no agreement between the results. In these cases, additional analysis must be performed for each entity. This simple example illustrates how to use different results to help the composition of a single final model. In many cases, the aggregation of results provide technical assistance, by highlighting the common patterns. It is possible to gain confidence that agreement across a collection of results can reflect the system structure [11].

However, a more careful analysis of the results, may reveal that other factors may influence the final model produced by clustering and visualization software techniques. A software system throughout its life cycle is susceptible to several changes in its architecture. However, such operations can introduce architectural violations in the code, for example, violation of the layers, break of abstractions, feature

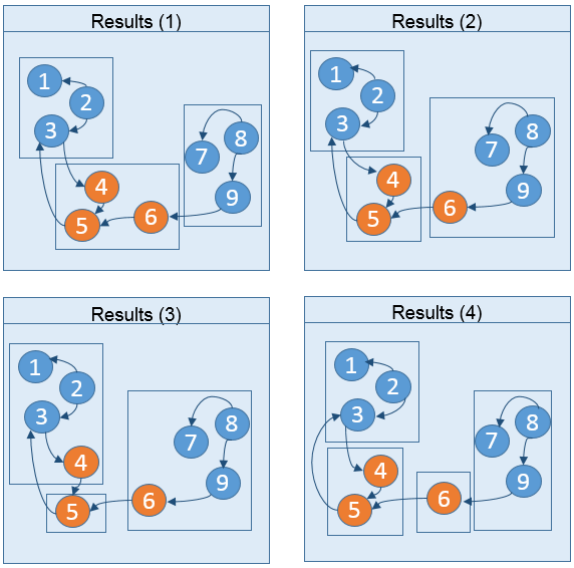

Figure 1: Different results for comparison.

duplication [7]. In this context, reverse engineering methods are strongly affected by those shortcomings in the system code base [13]. Such violations should be identified and addressed, so it does not affect the correctness of the final model.

To illustrate a violation on an architecture, we will use the same results of Figure 1. This Figure illustrates a representation of an architecture components through a dependency structure matrix. The figures depicts how the interaction works. The software elements are numbered from 1 to 4, where, for example, the Presentation module (2) requires information from the Visualization module (1). On the other hand, the figures shows also that the Visualization module (1) provides information to the Data module (4).

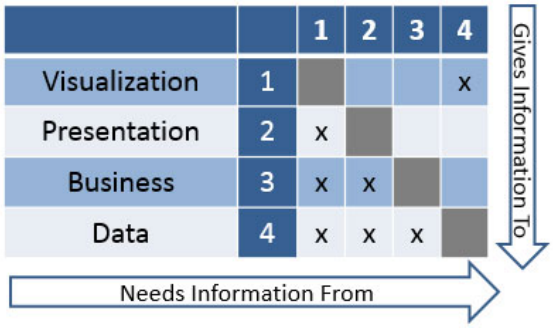

Figure 2: DSM representation of dependences.

Assuming the entity 4 is wrongly mapped as part of a module. This happens due to a coupling between entities 4 and 5, which illustrates an architecture violation. All results would be classified differently if such violation was removed, as shown in Figure 3. The results of the new analysis, taking into account the aggregation of similar entities, can be seen in Table 2. Analyzing the results, it is possible to check

Table 2: Occurrences of entities between the results after elimination of architectural violation

Relation	Occurrence
{1,2,3,4}	100%
{5}	50%
{5,6}	50%
{6}	25%
{6,7,8,9}	25%
{7,8,9}	100%

the impact of the violation. The module containing the first mapping {1,2,3} now adds entity 4, since this set was rated similarly on all results. As for the mapping {5,6} there is also a higher chance of being classified in a same module, as it occurs with higher frequency.

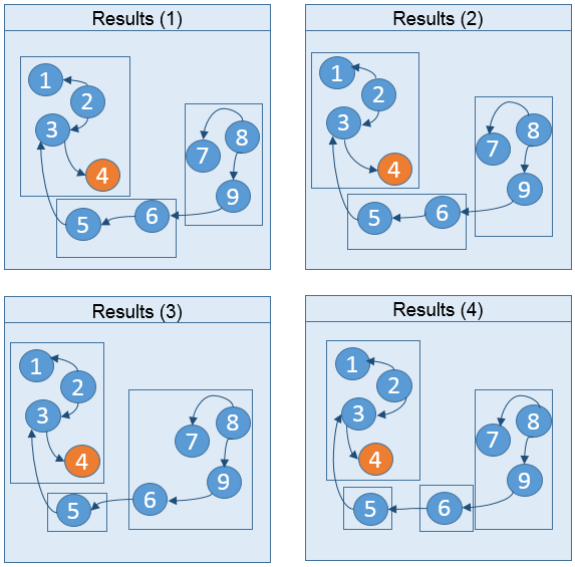

Figure 3: Eliminating the architectural violation of results.

Such violations can be found through an analysis of the artifacts produced by software visualization. As an example, take into account the dependency structure matrix shown previously in Figure 2. Through observation of the DSM, it is possible to note a relationship exists between layers, and each layer depends on the upper layers. However, this relationship is violated by layer "Date" (4), since it uses resources of the layer "View" (1), characterizing an architectural violation. Once the violation is detected, it should be treated as failure and, if necessary, modify the data set for the clustering process, so that the relationship is not considered. Thus, avoiding the production of wrong models that may affect the interpretation of the results.

4. EXPERIMENTAL STUDY

Empirical research presented in this experimental study uses a quantitative approach, in which the objective of the research is to obtain a numerical relationship between several variables or alternatives analysis. The goal of the experiment is to evaluate the use of software visualization and data clustering techniques in the context of a software architecture recovery.

4.1 Design

This section describes the experiment plan, and demonstrates how the experiment was designed. This allows the execution of other studies based on the same plan [6].

Figure 4 illustrates the experimental design. Subjects were divided into two groups. Participants in Group 1 started the extraction in a Visual Basic system using the clustering technique. The results of this extraction was collected. Then were presented the second technique and requested to produce a second model, applying both techniques. Afterwards, the same procedure was applied to the Java systems, but on the reverse order, i.e, first using software visualization and then clustering. For the participants in group 2, the same procedure applied, but starting with the software visualization technique.

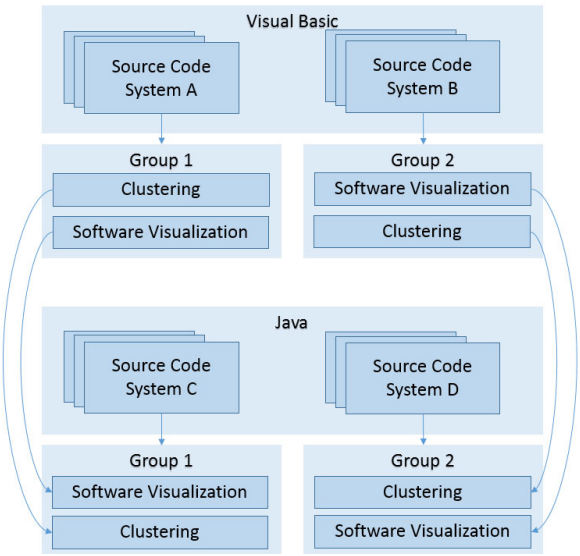

Figure 4: Design of the experiment.

4.1.1 *Experiment objects*

The study uses four software systems, written in two programming languages, as follows: Visual Basic and Java. Table 3 presents relevant information on the characteristics of each object of the experiment system. Systems A and B are legacy systems still operating in the University of Brasilia, and support the management of the academic routine of college students. Systems C and D have been developed on a new platform, in order to replace the systems written in Visual Basic. These systems deal with administrative university routines.

4.1.2 *Subject selection*

Participants in the experiment are system developers, with different levels of experience. A total of ten participants was selected. Subjects were divided into two groups randomly.

At the beginning of each session, participants were asked

Table 3: Systems objects used in the experiment.

System	Language	LOC	methods	files
System A	Visual Basic	25425	1551	133
System B	Visual Basic	36169	2828	218
System C	Java	19609	2699	195
System D	Java	14238	1734	178

about the level of knowledge in important aspects in a architecture recovery process. Figure 5 details the expertise of the participants in Group 1. The y-axis represents the percentage of professionals in the corresponding expertise level.

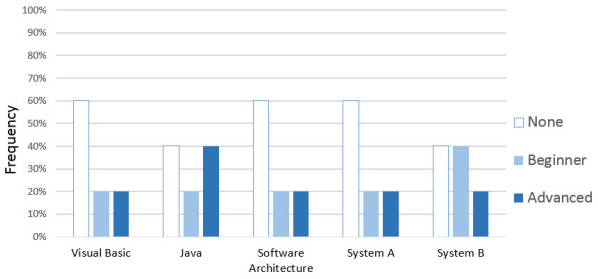

Figure 5: Participants expertise in Group 1.

Figure 6 details information about the level of knowledge of participants in Group 2.

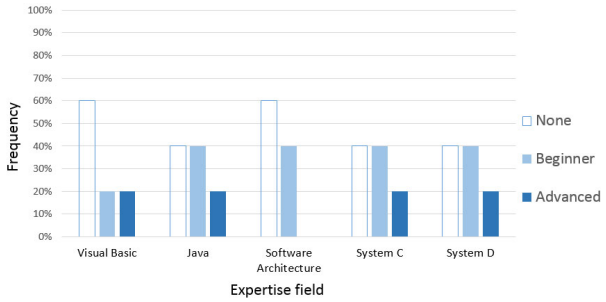

Figure 6: Participants expertise in Group 2.

4.2 Execution

Architecture recovery is an interpretive and interactive process involving many activities. Therefore, it may be a time consuming task. For this reason, the experiment seeks to recover the overall structure of the systems architecture. So, for each participant, the basic architecture concepts were introduced, as well as the essential elements for detecting the overall architecture of the object systems.

The analysis of the system object in Visual Basic using software visualization tool, involved the use of VBDepend tool. This tool provides several mechanisms that facilitates the exploitation of the system architecture in Visual Basic language based on visualization via dependency graphs and dependency structure matrix. The Figure 7 highlights a dependency structured matrix(DSM) extracted by the use of VBDepend tool. The DSM is useful for analyzing the properties of complex applications. In an architectural analysis, the couplings between the architectural modules are described and various operations on the matrix can be performed in order to identify architectural relations [16].

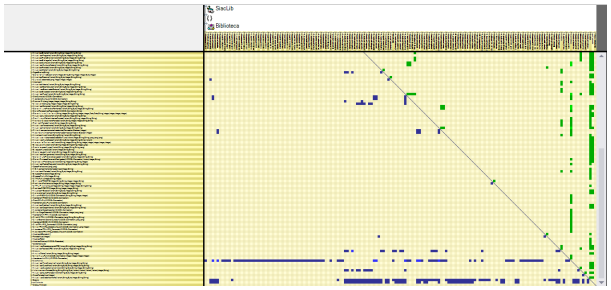

Figure 7: Dependency matrix obtained through the use of VBDepend tool.

To analyse Java systems, X-Ray and Architexa were used. The X-Ray software visualization tool is an open source software, available via plug-in for the Eclipse IDE. Through this tool, you can get visions of class-package dependencies and systemic complexity views on Java projects. An example of a systemic complexity view can be seen in 8. In this kind of view, classes are represented by rectangles; the width of the rectangles representing the number of methods implemented in a particular class, while the rectangle represents the number of lines of code in the class. The bonding edge is the relationship between the classes. The nodes can be set as a vertical tree, highlighting the hierarchy of classes.

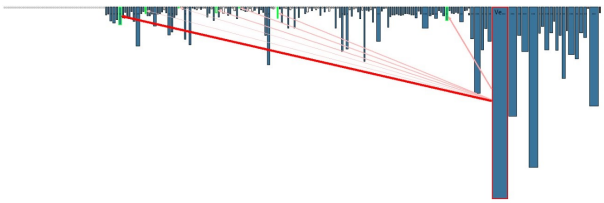

Figure 8: Extracted systemic complexity view of a system in Java.

Another tool used for viewing the source, and available to the participants of the experiment is the Architexa. With this tool it is possible to create different diagrams from the static analysis of the code. It is also available as a plug-in for the Eclipse IDE. Using this tool, the participants of the experiment can explore different aspects of the source code, such as a diagram of layers, as in Figure 9. The diagram layers groups classes based on their respective directories or packages and illustrates the dependency relationships between them. Also, different metrics can be used to highlight important aspects of the system, for example, the rectangle representing the software entities. Software entities that contain a large amount of code are represented by rectangles proportional to their size, which facilitates the identification of important aspects of the code structure. When selected a software entity, the visualization tool displays its respective dependencies by means of an arrow which indicates the origin and destination of the link. The thicker the arrow,

Table 4: Definition of the accuracy evaluation Software Architecture Recovery Techniques.

Purpose	Evaluate
Issue	accuracy of the models produced
Object	architecture recovery using visualization and clustering techinques
Viewpoint	software architect /application engineer
Context	University of Brasilia Academic Management Systems

the higher the correlation between the elements. Moreover, the colors of each rectangle are changed to emphasize the dependencies.

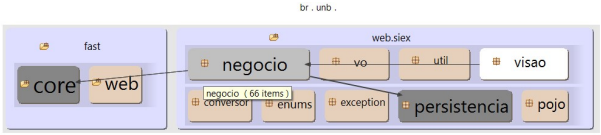

Figure 9: Extracted systemic complexity view of a system in Java.

The Bunch tool [12] was used for clustering approach as the tool is open source and has an intuitive graphical interface, as shown in Figure . Participants in the experiment were instructed to choose between the available algorithms and run the clustering process in the system under analysis. Participants can change the input parameters of the algorithms and run the clustering tool as often as they felt necessary. The process outputs is a file containing the modules automatically mapped by the selected algorithm. Through the output of the analysis, the participants comprised the architectural model of the system under analysis.

4.3 Analysis

The analysis was performed based on a plan Goal Question Metric (GQM), described in Table 4.

The underlying research questions the experiment are as follows:
RQ1: Does the use of software visualization technique, along with the clustering technique, increases the accuracy of the architectural model produced, in comparison with using only one of the techniques?
RQ2: Does the results are affected by the programming language used in the object systems?
RQ3: Can the order of execution of the techniques influence the results?

4.4 Metrics

To calculate the accuracy of the obtained architectural models, and answer the questions of the experiment, we used the Jaccard similarity coefficient, defined by the formula:

$$Sj = \frac{a}{a+b+c}$$

Here: Sj is the coefficient of Jaccard; a= the number of common modules; b= number of modules in B but not in A; c= number of modules in A, but not in B. In the experiment, model B was produced by a domain expert, while the model A was produced by participants in the experiment.

Each participant prepared two models per session. In the first model, the participant applied only one technique (or clustering software or software visualization). After obtaining the first model, the other technique was presented and it was requested to produce a new model, based on the two techniques. So, it was possible to calculate the similarity of model produced using only one technique and then compare the similarity of model produced using the two techniques.

4.5 Results

Through the results, calculated from the comparison of the models produced by the participants with models produced by experts in the field, it was possible to investigate the use of architecture recovery approach using clustering and software visualization techniques.

To analyze the significance of the results, it was used the paired t-test. For this, we observed the different conditions that underpin the paired t-test, as: two related groups; no significant outliers; distribution of the differences in the dependent variable between the two related groups is approximately normally distributed. Table 5 describes the comparison data between the scores obtained using one and two techniques. It is possible to see that the average accuracy of the models obnained in Visual Basic sytems using one technique was approximately 51%, and the average accuracy of the models obtained using two techniques was approximately 77%. A difference of 26%. On the other hand, is possible to note that the mean of the models obtained in Java systems using one technique was approximately 48%; when used two techniques, the mean was approximately 72%. A difference of 24%.

Table 5: Descriptive data table of the systems in Visual Basic and Java.

	Technique	N	Means	Std. Deviation	Std.Error Mean
Visual Basic	One Technique	10	0.517	0.091	0.028
	Two Techniques	10	0.771	0.077	0.024
Java	One Technique	10	0.483	0.060	0.019
	Two Techniques	10	0.728	0.059	0.018

We now need to analyse if these results are statistically meaningful. We carry out the paired t-test then. Table 6, details the paired t-test to compare the results obtained using one technique followed by two techniques. To interpret the results is necessary to observe the Sig. (2-tailed) value, also known as p value. This is the value that determines whether the means are statistically different. If the Sig.(2-tailed) value is greater than 0.05 then there is no statistically significant difference between the two conditions. On the other hand, if the value is less than or equal to 0.05 then there is a statistically significant difference between the two conditions. The significance level of 0.05 reflects the 95% confidence interval used as parameter value for the calculations.

Analyzing the data for the systems in Visual Basic, it is possible to conclude that there is a significant difference between the scores obtained using one technique (Mean= 0.517, Std. Deviation = 0.091) and two techniques (Mean= 0.771, Std. Deviation= 0.077); t(9)=-10.046, p= 0.000. So, with 95% confidence, it can be said that there is an increase in results from 0.196 to 0.310 when used two techniques for software

architecture recovery. Analyzing the data for the systems in Java language, it is possible to conclude that there is a significant difference between the mean values obtained using one technique (Mean= 0.483, Std. Deviation= 0.060) and two techniques (Mean= 0.728, Std. Deviation= 0.059); t(9)= -10.046, p= 0.000. Thus, with 95% of confidence, the accuracy improvement varies from 0.207 to 0.283 when used two techniques for software architecture recovery. For the sistems in Java

Table 6: Paired t-test for Visual Basic systems.

	Paired Diferences					t	df	Sig. (2-tailed)
	Mean	Std. Deviation	Std. Error Mean	95% Confidence Interval Lower	Upper			
(Visual Basic) One technique-Two techniques	-0.253	0.079	0.025	-0.310	-0.196	-10.046	9	0.000
(Java) One technique-Two techniques	-0.245	0.053	0.0169	-0.283	-0.207	-10.046	9	0.000

After these analyzes, it is possible to conclude that using two techniques produced better results than using only one technique in a software architecture recovery process. Thus, it is possible to answer RQ1, where it is questioned whether the use of both techniques increases the accuracy of the obtained architectural models. Given the above results, it is possible to say that the use of both techniques had a positive impact on the accuracy of the models produced, since all the results indicated an improvement in the accuracy of the models when used the two techniques together.

To answer RQ2, we performed an analysis of the impact on the programming language on the accuracy of the results. From Table 7 it can be seen that in Visual Basic systems, the average accuracy of the models produced was approximately 60%. In systems written in Java, the accuracy was approximately 64%. These values represent the total amount of analyses performed for each language. Since each language was twice analysed in each group, then N equal 20, in this case.

Table 7: Descriptive data table comparing system languages.

	Language	N	Mean	Std. Deviation	Std. Error Mean
Pair1	Visual Basic	20	0.605	0.138	0.031
	Java	20	0.644	0.153	0.034

To calculate the significance of the means difference and determine if the programming language affects the recovery process, we conducted a paired t-test. The result can be seen in Table 8. Through data analysis, we conclude that there is no significant difference between the mean values of the systems in Visual Basic (Mean= 0.605, Std. Deviation= 0.138) and Java systems (Mean= 0.644, Std. Deviation= 0.153); t(19)= -1.867, p= 0.077. Thus it is possible to answer RQ2: given the outcome, we can conclude that, with 95% of confidence level, the programming languages had no statistically significant effect on the results, since Sig(2-tailed) value is greater than 0.05. So, the difference of means is in the interval (-0.082; 0.004), which includes 0.

Finally, it is possible to analyze if the order of the recovery technique affects the results. This analysis is useful to

Table 8: Paired t-test for the comparison of the two programming languages.

	Paired Diferences					t	df	Sig. (2-tailed)
	Mean	Std. Deviation	Std. Error Mean	95% Confidence Interval Lower	Upper			
Visual Basic-Java	-0.0387	0.0928	0.020	-0.082	0.004	-1.867	19	0.077

check whether starting from visualization or clustering in our process influences the accuracy of the results. Table 9 details the descriptive results of the data. When using software visualization and then using clustering, the accuracy of the models obtained was approximately 77%. Moreover, using first the clustering technique then software visualization technique, the accuracy of the models obtained was approximately 72%.

Table 9: Descriptive data table related to the order of the techniques.

	Order	N	Mean	Std. Deviation	Std. Error Mean
Pair1	Visualization-Clustering	10	0.771	0.077	0.024
	Clustering-Visualization	10	0.728	0.059	0.018

To check for the difference between the means, it was conducted a paired t-test. The results are shown in Table 10. Through data analysis, it is possible to conclude that there is no significant difference between the mean values obtained using Software Visualization prior to Clustering (Mean=0.771, Std. Deviation= 0.077) and Clustering prior to Software Visualization (Mean= 0.728, Std. Deviation= 0.059); t(9)= 1.456, p= 0.179.

Table 10: Paired t-test for the comparison of the order of techniques.

	Paired Diferences					t	df	Sig. (2-tailed)
	Mean	Std. Deviation	Std. Error Mean	95% Confidence Interval Lower	Upper			
VisualizationClustering-ClusteringVisualization	0.0427	0.0927	0.029	-0.023	0.109	1.456	9	0.179

Given the above, it is possible to answer RQ3, whether the order of the technique applied can affect the results. Facing the results, we can say that there is no statistical evidence to say that the order of the approach had an influence on the results, since Sig(2-tailed) value is greater than 0.05 resulting the interval (-0.023, 0.109) which includes 0.

4.6 Internal validity

All participants of the experiment have experience in analysis and development of software systems, which contributes to the representation of the group of analysts and system developers who may require an architectural reconstruction process. The participants of the two experimental groups have similar characteristics.

The use of two programming languages, Visual Basic and Java, contributes to the representation of an organizational environment in which there is no homogeneity in relation to the programming language used by the systems in the company. Despite the fact that the participants are separated into two groups and run the experiment in different objects

systems, recovery approach was conducted in systems with similar architectures. Due to this similarity, it is expected that the results are not influenced by the programming language objects, or difference between the systems.

4.7 External validity

The size in LOC and the complexity of the systems objects can influence the generalization of the results. However, such systems present complexity and size similar to most of the systems developed to meet sectorial needs of an organization.

Also, the limited number of participants of the experiment can not allow a generalization of the experiment. However, through the separation method of the group at random, it was expected to decrease the confusion factor.

4.8 Discussion

The experimental study observed the impact of using data clustering and software visualization techniques in the understanding of software systems architecture.

First, it was examined whether the use of software visualization and clustering techniques together improves the accuracy of the architecture recovery process. The results showed that, when used one after the other technique, the accuracy of the models produced had a significant increase. This is because the techniques act in a complementary manner by providing additional information. Also, the use of both techniques together provided to the participants a different perspective of the software, not perceived or not represented when compared to using only a technique. Thus, allowing for recovery of the architecture more accurately.

Second, an analysis was performed to verify if programming language can affect the recovery of the architecture. The results showed that the programming language had no significant effect on the results. This demonstrates the flexibility of clustering and software visualization techniques. This flexibility is due to the fact that currently there are several software visualization tools available, which enable the analysis of different programming languages. Also, the clustering approach, implemented by Bunch tool, is designed to be independent of language, since it is only necessary to set up a MDG containing the dependency relationships between the entities of the source code, and this is achieved through static analysis techniques of the source code.

Finally, we observed that the order of execution of the techniques does not significantly affect the results. This indicates that there is no specific order for the application of the techniques in the approach. This was also confirmed by the feedback from participants. When asked about the preference of the order of execution of techniques, some preferred to start with clustering, while others, with the visualization. Thus, the execution of techniques can happen in a subjective way that best meets the needs of the responsible for the recovery of architecture.

5. CONCLUSION

One of the factors that can influence the success or failure of a process of legacy system modernization is the understanding of its architecture. In this sense, the time taken to recover these concepts is as important as the time spent in planning the new system. This is due to the fact that, for a complete understanding of a legacy system, the first step is to understand its architecture, because this is the base that supports all system features. Thus, for the architecture recovery process as complete and accurate as possible, it is essential the use of different techniques.

In this context, the use of software visualization techniques for analysis and recovery of a system architecture is essential, since it allows greater flexibility to the process of modernization. Through this technique, it is possible to obtain a compact representation of the entire structure of the source code. The various aspects of the software implemented in several lines of code may be represented by a single diagram, which summarizes all this complexity.

In addition, an architecture recovery process using clustering techniques is also effective. A representation automatically created by a data collation process is a quick and convenient way to explore complex systems, often totally unknown to the analyst. This is is a good first step to understanding important aspects of the software. In other cases, the clustering process can provide different perspectives in understanding of software functionalities.

To some extent, our work is the first of a kind where we explore the use of both techniques, the visualization and clustering software, to provide a wide range of system representations analysis. Experiment performed our approach can significantly improve the architecture process disregard the programming language or the order of the techniques applied. We believe such representations will allow different views on various aspects of the software, which contributes to the understanding of the whole system structure. Thus, the use of these techniques allows for the recovery of a system architecture, with agility and accuracy.

For future work, we plan to make a comprehensive analysis on public software repository, e.g. GitHub, and conduct a more thorough empirical study, also considering other programming languages like C or C++.

6. REFERENCES

[1] G. Canfora, M. Di Penta, and L. Cerulo. Achievements and challenges in software reverse engineering. 54(4):142, 2011.

[2] S. Ducasse and D. Pollet. Software architecture reconstruction: A process-oriented taxonomy. 35(4):573–591, 2009.

[3] L. Feijs and R. De Jong. 3d visualization of software architectures. *Commun. ACM*, 41(12):73–78, Dec. 1998.

[4] Y. Ghanam and S. Carpendale. A survey paper on software architecture visualization. 2008.

[5] A. K. Jain, M. N. Murty, and P. J. Flynn. Data clustering: A review. *ACM Comput. Surv.*, 31(3):264–323, Sept. 1999.

[6] N. Juristo and A. M. Moreno. *Basics of Software Engineering Experimentation*. Springer Publishing Company, Incorporated, 1st edition, 2010.

[7] R. Kazman and S. J. Carriere. View extraction and view fusion in architectural understanding. In *Software Reuse, 1998. Proceedings. Fifth International Conference on*, pages 290–299. IEEE, 1998.

[8] M. Lanza. Codecrawler-lessons learned in building a software visualization tool. In *Software Maintenance and Reengineering, 2003. Proceedings. Seventh European Conference on*, pages 409–418. IEEE, 2003.

[9] M. Lungu and M. Lanza. Exploring inter-module relationships in evolving software systems. In *Software Maintenance and Reengineering, 2007. CSMR '07. 11th European Conference on*, pages 91–102, March 2007.

[10] S. Mancoridis, B. S. Mitchell, Y. Chen, and E. R. Gansner. Bunch: A clustering tool for the recovery and maintenance of software system structures. In *Proceedings of the IEEE International Conference on Software Maintenance*, ICSM '99, pages 50–, Washington, DC, USA, 1999. IEEE Computer Society.

[11] B. Mitchell and S. Mancoridis. Craft: a framework for evaluating software clustering results in the absence of benchmark decompositions [clustering results analysis framework and tools]. In *Reverse Engineering, 2001. Proceedings. Eighth Working Conference on*, pages 93–102, 2001.

[12] B. S. Mitchell. A heuristic search approach to solving the software clustering problem. 2002.

[13] M. Platenius, M. von Detten, and S. Becker. Archimetrix: Improved software architecture recovery in the presence of design deficiencies. In *Software Maintenance and Reengineering (CSMR), 2012 16th European Conference on*, pages 255–264, March 2012.

[14] N. Sangal, E. Jordan, V. Sinha, and D. Jackson. Using dependency models to manage complex software architecture. In *Proceedings of the 20th Annual ACM SIGPLAN Conference on Object-oriented Programming, Systems, Languages, and Applications*, OOPSLA '05, pages 167–176, New York, NY, USA, 2005. ACM.

[15] M. Shtern and V. Tzerpos. Clustering methodologies for software engineering. *Adv. Soft. Eng.*, 2012:1:1–1:1, Jan. 2012.

[16] B. Tekinerdogan, F. Scholten, C. Hofmann, and M. Aksit. Concern-oriented analysis and refactoring of software architectures using dependency structure matrices. In *Proceedings of the 15th Workshop on Early Aspects*, EA '09, pages 13–18, New York, NY, USA, 2009. ACM.

[17] A. Teyseyre and M. Campo. An overview of 3d software visualization. *Visualization and Computer Graphics, IEEE Transactions on*, 15(1):87–105, Jan 2009.

[18] V. Tzerpos and R. Holt. Accd: an algorithm for comprehension-driven clustering. In *Reverse Engineering, 2000. Proceedings. Seventh Working Conference on*, pages 258–267, 2000.

[19] V. Tzerpos and R. C. Holt. Acdc: An algorithm for comprehension-driven clustering. In *Proceedings of the Seventh Working Conference on Reverse Engineering (WCRE'00)*, WCRE '00, pages 258–, Washington, DC, USA, 2000. IEEE Computer Society.

[20] R. Wettel, M. Lanza, and R. Robbes. Software systems as cities: a controlled experiment. In *Proceedings of the 33rd International Conference on Software Engineering*, pages 551–560. ACM, 2011.

[21] T. Wiggerts. Using clustering algorithms in legacy systems remodularization. In *Reverse Engineering, 1997. Proceedings of the Fourth Working Conference on*, pages 33–43, Oct 1997.

[22] V. Zapalowski, I. Nunes, and D. J. Nunes. Revealing the relationship between architectural elements and source code characteristics. In *Proceedings of the 22nd International Conference on Program Comprehension*, pages 14–25. ACM, 2014.