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

Renato Paiva, Genaína N. Rodrigues, Marcelo Ladeira, Rodrigo Bonifácio
Computer Science Department
University of Brasília
Brasilia, Brazil
{renatoedesio, mladeira@unb.br},
{genaina, rbonifacio}@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 quasi-automatic and quasei-manual techniques for architecture recovery have been proposed and extensively used. However, there are still important improvements that need to be addressed on this arena.~~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

Software modernization is an activity that requires a significant understanding of the architecture. Nevertheless, this is a challenge since there is usually a myriad of absent architecture information due to the lack of architecture specification documents, particularly w.r.t legacy systems. As a result, some reverse engineering practices must take place to reconstruct the architectural structure of a software system. Moreover, the ability to mine relevant information from execution traces tend to quickly become large and unmanageable. For this reason, it is necessary to filter part of those traces and extract relevant information for the particular understanding of the task being performed [1]. As a result, the study of methods and tools to extract the software systems architecture is fundamental to guarantee a fast and accurate process. ~~In most cases, understanding the software structure is also 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 is out-of-date, a software architecture recovery process must take place.~~

Software visualization is a widely used technique for architecture recovering [3,5,9,15]. They are *semi-automatic* techniques that reconstruct the software architecture by manually abstracting low-level knowledge, due to interactive and expressive visualization tools [2]. The basis of the software visualization underlies on the creation of a representation of the system via visual elements [15]. 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.

Despite the improvement on the results obtained through semi-automatic techniques~~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,particularly to derive the modular decomposition of the software, 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 recovery process. For instance, software clusterization is an automatic technique to recover a software system architecture [6,11,13,17]. In the context of architecture recovery, clustering techiniques aims at grouping software artifacts in significant modules, leading towards an understanding of the software system structure in a higher abstraction level [13]. However, Lutellier *et al.* [**?**] has recently pointed out that, apart from the selection of the architecture recovery techniques, the accuracy was low for all studied techniques, corroborating past results [4].

On the other hand, although both approaches for architecture recovery have been widely discussed in the literature, there is no empirical study that investigates how each technique complements each other with the purpose of achieving a more accurate architecture recovery process. In this paper we carried out an experiment that aims at understanding how the combination of both approaches improve the accuracy of the models obtained from an architectural recovery. The contributions of the paper are twofold:

- It presents the design, execution, and main findings of an empirical study that investigates the benefits of combining well-known semi-automatic visualization and clustering techniques for architecture recovering.

- Without loss of generality w.r.t the recovery process applied, it reports that the combination of the selected approaches can improve the accuracy of the resulting models from 19% to 30%.~~ that improvement ranges from 19% to 30%, depending on the visualization and programming language techinique we used in our experiment.~~

The remaining sections of the paper are structured as follows: Section 2 presents related work regarding semi-automatic and automatic~~software architecture visualization and clustering~~ techniques. Section 3 highlights the inaccuracies of such techniques and the principles that drive our investigation. Section **??** is the core 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 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.

Garcia *et al.* [4] performed a comparative analysis of six automated architecture recovery techniques. The selected techniques rely on two kinds of input obtained from implementation-level artifacts: textual and structural. The accuracy of the techniques were asses on eight architectures from six different open-source systems. The results obtained indicate that two of the selected recovery techniques are superior to the rest along multiple measures. However, the results also show that there is significant room for improvement in all of the studied techniques.

Lutellier *et al.* [**?**] compared nine variants of six architecture recovery techniques using two different types of dependencies: symbol and include. Four of the selected techniques use dependencies to determine clusters, while the remaining two techniques use textual information from source code. The results shows that symbol dependencies generally produce architectures with higher accuracies than include dependencies. Despite this improvement, the overall accuracy is low for all recovery techniques.

Regarding the use of semi-automatic techniques for understanding software systems, Wettel *et al.* [16] 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.

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 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 [10], 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

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%

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 [10].

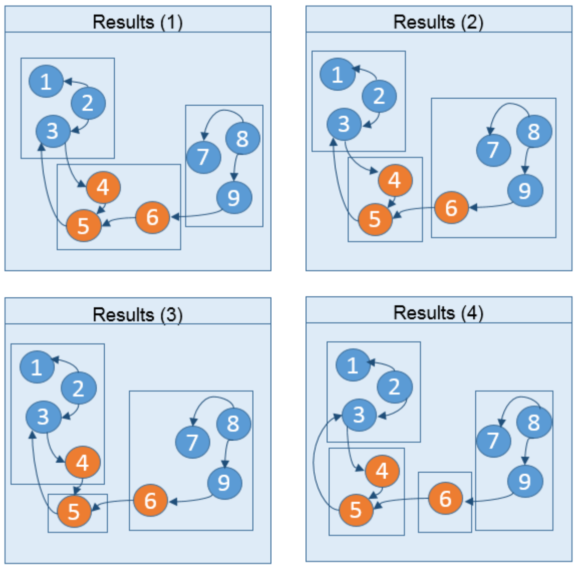


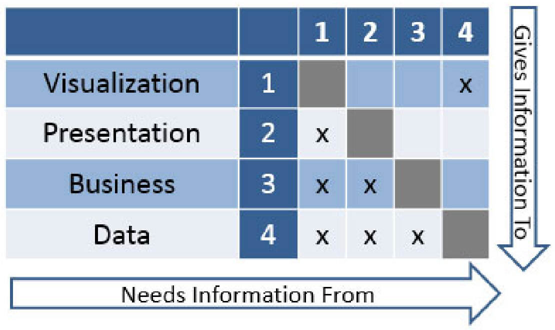
Figure 1: Different results for comparison.

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 duplication [8]. In this context, reverse engineering methods are strongly affected by those shortcomings in the system code base [12]. Such violations should be identified and addressed, so it does not affect the correctness of the final model.

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%

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).



		1	2	3	4
Visualization	1	■			x
Presentation	2	x	■		
Business	3	x	x	■	
Data	4	x	x	x	■

Gives Information To
Needs Information From

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 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.

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,

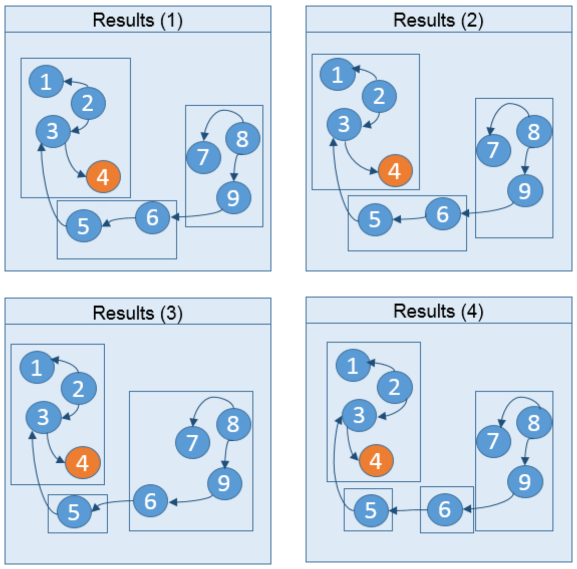


Figure 3: Eliminating the architectural violation of results.

Table 3: Definition of the accuracy evaluation Software Architecture Recovery Techniques.

Purpose	Evaluate
Issue	accuracy of the modular decomposition
Object	architecture recovery using semi-automatic and automatic techinques
Viewpoint	software architect / application engineer
Context	University of Brasilia Academic Management Systems

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

The empirical ~~Empirical~~ research presented in this ~~experimental~~ aims to evaluate the combination of well-known semi-automatic (software visualization) and automatic (data clustering) techniques in the context of a software architecture recovery to understand the modular decomposition of a system. We guided our study~~The study was performed~~ using the Goal Question Metric (GQM) approach, as described in Table 3.

The underlying research question for this experiment is as follows: *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?*

Each participant recovered the architecture by applying two reengineering approaches per session, which produced two models representing de modular decomposition of the system. In the first session, the participant applied only one technique using either a semi-automatic decomposition based on software visualization or an automatic decomposition based on software clustering. After obtaining the first model, the other approach was presented and it was requested to produce a new model, based on the two techniques. In this way it was possible to calculate the accuracy of the resulting models.

To calculate the accuracy of the recovered architectural representation of the modular decomposition of the systems, and then answering 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 receovered modules; $b=$ number of modules recovered in B but not in A; $c=$ number of recovered 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 theparticipants of~~in~~ the experiment.

4.1 Design

This section presents the experimental design of this research, in a level of details that might help other researchers to replicate our study [7] using a similar setting.

The study considered four software systems, written in two programming languages (Visual Basic and Java). Table 4 presents relevant information on the characteristics of each object of the experiment. Systems A and B are legacy systems still operating in the University of Brasilia, and both support the management of the academic routine of college students. Systems C and D have been developed on a new platform, in order to modernize legacy systems written in Visual Basic. These systems deal with administrative university routines.

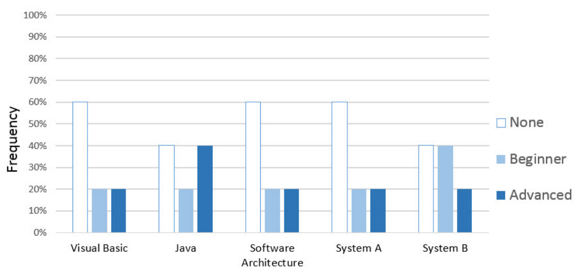
Table 4: Systems objects used in the experiment.

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

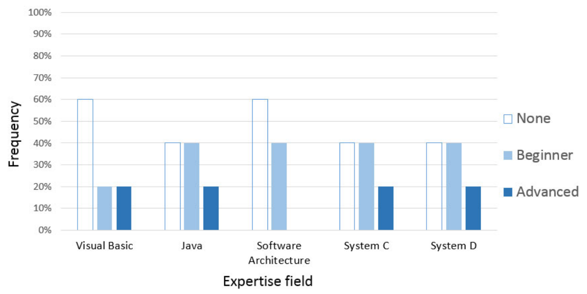
The participants involved in the experiment are system developers, with different levels of experience. We selected a total of ten participants using a convenience approach, that is, the participants volunteered for participating in the study. Subjects were divided into two groups randomly. At the beginning of each session, participants were asked about their level of knowledge in relevant aspects related to architecture recovery. Figure 4(a) and Figure 4(b) detail the expertise of the participants in the first and second groups, respectively. The y-axis represents the percentage of professionals in the corresponding expertise level.

As explained[1], the subjects were divided into two groups. Participants in Group 1 started the extraction in a Visual

[1]seria importante descrever quais são as variáveis envolvidas, e quais fatores precisam / estão sendo controlados com esse design. como os grupos foram formados? talvez uma leitura do artigo SoSyM ajude.



(a) Participants expertise in Group 1.



(b) Participants expertise in Group 2.

Figure 4: Expertise of the participants.

Basic system using the clustering technique. The results of this extraction was collected. Then we presented the second technique (a semi-automatic visualization tool) and requested the subjects 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 a software visualization tool and then clustering. For the participants in Group 2, the same procedure applied, but starting with the software visualization technique.

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 modular decomposition 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 semi-automatic analysis of the Visual Basic system was carried out using the `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. Figure 5 highlights a dependency structured matrix (DSM) [14] obtained using `VBDepend` tool.

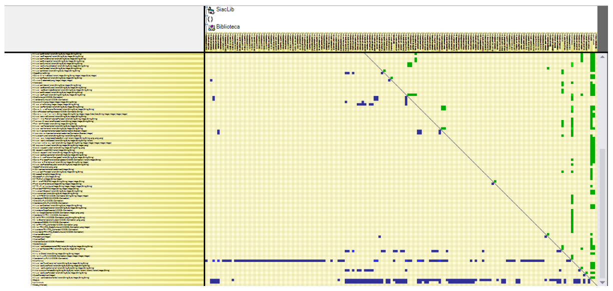


Figure 5: Dependency matrix obtained through the use of VBDepend tool.

To analyse the Java systems, the subjects used two different tools: `X-Ray` and `Architexa`. The X-Ray software visualization tool is an open source software available as a plug-in for the Eclipse IDE. Through this tool, one can get different visions of class-package dependencies and systemic complexity views on Java projects. An example of a systemic complexity view can be seen in 6. 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 height of 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.

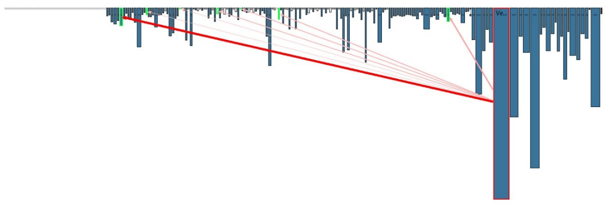


Figure 6: Extracted systemic complexity view of a system in Java.

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

The `Bunch` tool [11] was used as representant of the automatic approach for architecture recovery. It is an open-source initiative that implements a recovering approach based on clustering provides an intuitive graphical interface, as shown in Figure ??. Participants in the experiment were instructed to choose between the available algorithms and run

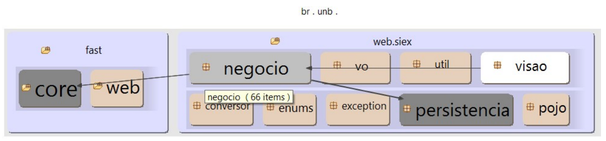


Figure 7: Extracted systemic complexity view of a system in Java.

the clustering process in the system under analysis. The participants were able to change[2] the input parameters of the algorithms and run the clustering tool as often as they felt necessary. `Bunch` generates a file containing the modules automatically mapped from the selected algorithm. Through the output of the analysis, the participants comprised the architectural model of the system under analysis.

4.3 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 approaches using a combination of automatic (clustering) and semi-automatic (software visualization) techniques to identify the modular decomposition of a system.

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 presents the comparison data between the scores obtained using one and two techniques. It is possible to realize 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 the combination of both techniques was approximately 77%— that is, a difference of 26%. On the other hand, it is possible to note that the mean of the models obtained in Java systems using one technique was approximately 48%; when used both techniques, the mean was approximately 72%— a difference of almost 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 also analysed whether these results are statistically meaningful, carring out the paired `t-test`. Table 6 details the results of the test, comparing the results obtained using one technique with the outcomes obtained using two techniques (automatic and semi-automatic). To interpret the results it is necessary to observe the Sig. (2-tailed) value, also known as `p-value`. 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, we can assume 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.

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

	Paired Diferences					t	df	Sig. (2-tailed)
				95% Confidence Interval				
	Mean	Std. Deviation	Std. Error Mean	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 our research question. Given the above results, we conclude that the use of both techniques present 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.

We also analyzed if the order of the recovery technique affects the results. This analysis is useful to check whether starting from visualization or clustering in our process influences the accuracy of the results. Table 7 details the descriptive results of the data. When first using software visualization and then using clustering, the accuracy of the models obtained was approximately 77%. Differently, using first the clustering technique then software visualization technique, the accuracy of the models obtained was approximately 72%.

Table 7: 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 investigate the difference between the means, it was conducted a paired `t-test`. The results are shown in Table 8. 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) or Clustering prior to Software Visualization (Mean= 0.728, Std. Deviation= 0.059); t(9)= 1.456, p= 0.179.

4.4 Discussion

[2]isso nao diminui o controle do experimento? isso nao diminui as chances de replicar o estudo?

Table 8: Paired t-test for the comparison of the order of techniques.

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

Altogether, the results showed that, when used one technique after the other, the accuracy of the models produced had a significant increase. This suggests that the investigated 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.

In addition, 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 the 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.

4.5 Threats to validity

Regarding internal validity, our study is limited to the choices we made with respect to the tools used in the experiment (for representign both semi-automatic and automatic approaches for architecture recovery) and the object systems used in the analyses. It is important to note that the selected tools have been widely used either in industry (such as `VBDepend` and `Architexa`) or in achademic efforts (`Bunch`).

Some criticism to our paper might also arise due to the the participants selection. Nevertheless, 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. Regarding the object systems, they are implementeed using two programming languages, Visual Basic and Java, which 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, the 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.

Regarding 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 that are expected to be similar to other 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.

5. CONCLUSION

To make the architecture recovery process as complete and accurate as possible it might be essential the application of different analysis techniques. Previous studies have shown that the overall accuracy in the architecture recovery process using a recovery technique alone is still low.~~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, to make the architecture recovery process as complete and accurate as possible it might be essential the application of different analysis techniques. Previous studies have shown that the overall accuracy in the architecture recovery process using a recovery technique alone is still low.~~

~~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 automatic and semi-automatic techniques, e.g. clustering and visualization, to jointly provide a more comprehensive as well as accurate architecture recovery. In this work, we make evidence for such claim in a industrial environment where our approach have significantly improved the architecture recovery process disregard the programming language under study (in this case Java and Visual Basic) or the order of the analysis 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 can potentially allow 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 with the purpose of analysing not only other projects developed in other programming languages like C or C++ but also the scalability of our approach.

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] J. Garcia, I. Ivkovic, and N. Medvidovic. A comparative analysis of software architecture recovery techniques. In E. Denney, T. Bultan, and A. Zeller, editors, *2013 28th IEEE/ACM International Conference on Automated Software Engineering, ASE 2013, Silicon Valley, CA, USA, November 11-15, 2013*, pages 486–496. IEEE, 2013.

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

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

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

[8] 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.

[9] 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.

[10] 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.

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

[12] 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.

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

[14] 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.

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

[16] 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.

[17] 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.