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

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. For this reason, semi-automatic and automatic techniques for architecture recovery have been active focuses of research. However, there are still important improvements that need to be addressed on this field of research w.r.t achieving a more accurate architecture recovery process. In this work, we propose 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 in a industrial environment to empirically evaluate our investigation in four commercial systems. Our results indicated a statistically significant increase in the accuracy of the recovered architectures in all cases.

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].

Software visualization is a widely used technique for architecture recovery [3, 5, 8, 16]. They are *semi-automatic* techniques that reconstruct the software architecture by manually abstracting low-level knowledge, due to interactive and expressive visualization tools [2]. Nevertheless, the accuracy of the results to derive the modular decomposition of the software from semi-automatic techniques is usually bound by the subjective ability of the analysts. As a result, various research work have been devoted to automate the architecture recovery process. To that extent, software clusterization has increasingly gained attention as an *automatic* technique to recover a software system architecture by grouping software artifacts in significant modules [6, 12, 14, 19].

However, recent work by Lutellier *et al.* [9] 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 semi-automatic and automatic approaches for architecture recovery have been widely discussed in the literature, there is no empirical study that investigates if and 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 in a industrial environment whether the accuracy of the models obtained from an architectural recovery can be improved by combining both automatic and semi-automatic techniques. Therefore, 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 recovery.
- Without loss of generality w.r.t the recovery process applied, it reports that the combination of the selected approaches together provided a surplus on the accuracy of the resulting models from 19% to 30% compared to the use of either of the techniques alone.

The remaining sections of the paper are structured as follows: Section 2 presents literature work most related to the focus of our work.. Section 3 highlights the inaccuracies of such techniques and the principles that drive our investi-

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

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

gation. Section 4 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 future directions we envision for our work.

2. RELATED WORK

In this section, due to space scarcity, we report only most recent work that leveraged the discussion comparing accuracy and effectiveness of recovery techniques.

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 assessed on eight architectures from six different open-source systems. Their results indicated 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, particularly w.r.t accuracy.

Lutellier *et al.* [9] 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 context of semi-automatic techniques, Wettel *et al.* [18] 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 analysis via CodeCity 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 manually.

3. RATIONALE OF THE APPROACH

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 to correctly interpret the 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 techniques, have 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 algorithms used during the process. Through the analysis of different models it is possible to identify patterns that recur frequently in the results. One of the most prominent software clustering tool is Bunch [12], which transforms the architecture recovery problem into an optimization problem. Bunch uses hill-climbing and genetic algorithms to find the software partitions that maximizes the modularization quality.

However, when the idea is not clear yet of how the system structure is composed, the various models produced by clustering process may not suffice [11], since the automatic technique might capture only a high abstraction level of the architecture recovered. In this sense, the use of software visualization techniques allow a fine-grained 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 those produced by the software visualization process, it is possible to get different representations of the system to form a final model with greater precision.

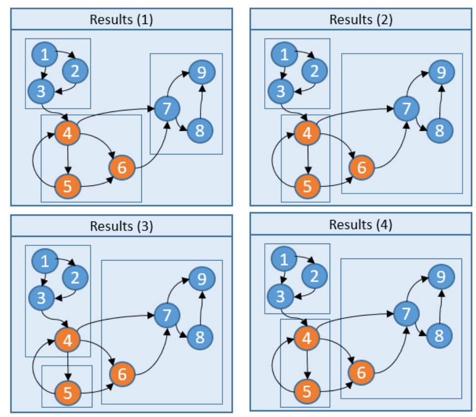


Figure 1: Different results for comparison.

We illustrate how such approach can be performed by taking 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 techniques and software clustering algorithms. Each model features nine entities, namely: {1,2,3,4,5,6,7,8,9}. The ratio of the frequency the entities were classified as part of the same group is observed in Table 1.

Table 1: Occurrences of entities in the results.

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

By analyzing the results in Table 1, it is possible to see that entities {1,2,3} can be classified into a single module, since they appear 100% of the times in the same cluster; so do the entities {7,8,9}. 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]. On the other hand, entities {4,5,6} may be classified either as a single module or scattered into different modules, since there is no agreement between the results. In such situations, additional analysis must be performed.

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 or feature 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 the correctness of the final model should not be affected.

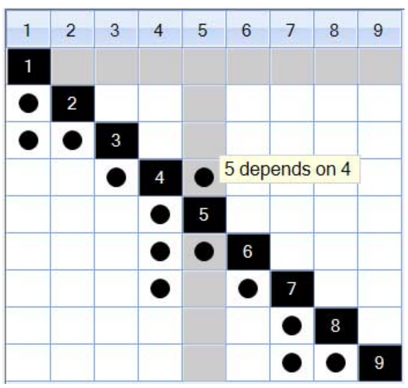


Figure 2: DSM representation of dependences.

To illustrate an architecture violation, we will use the results of Figure 2, which depicts a representation of architecture components through a dependency structure matrix (DSM). The figure depicts how the interaction works. The software elements are numbered from 1 to 9, where, for example, module 1 requires information from module 2. On the other hand, the figure shows also that module 4 provides information to module 5. This relationship is violated by module 5, since it uses resources of module 4, characterizing, thus, an architectural violation.

Table 2 presents the results of the new analysis, taking into account the aggregation of similar entities after the identified architecture violation was removed. 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 in 100% of the times as part of the same module. As for the mapping {5,6} there is also a higher chance of being classified in a same module, as both entities now occur as part of the same module with higher frequency after the architecture violation was detected and removed.

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

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

By these means, we propose to promote how practitioners may obtain more accurate architecture recovery process by interweaving complementary levels of abstraction provided by architecture recovery techniques. If a higher level of abstraction is needed, software clustering via Bunch tool, for exemple, allows the user to choose the number of clusters of the recovered architecture. This information could then be refined and verified by applying semi-automatic techniques, provided by tools like `VBDepend` [17], `Architexa` [15] or `X-Ray` [10]. On the other hand, we claim that the order of which technique to start with should not influence the accuracy of the architecture recovery due to the complementary level of information each technique provides. In the next section, we empirically evaluate not only such claim, but also the validity of our proposal.

4. EXPERIMENTAL STUDY

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

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

Purpose	Evaluate
Issue	architecture recovery accuracy
Object	semi-automatic & automatic techniques
Viewpoint	software architect & application engineer
Context	industrial environment

The underlying research question for this experiment is as follows: *Does the use of software visualization technique, along with the clustering technique, increase the accuracy of the architectural model produced, in comparison with using one of the techniques alone?*

4.1 Design

The study considered four software systems, written in two distinct 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.

The participants involved in the experiment are system developers, with different levels of experience. We selected a total of ten participants that agreed to voluntarily participate in the study. Subjects were randomly divided into two groups, henceforth called Group 1 and Group 2. At the beginning of each session, participants were asked about their level of knowledge in relevant aspects related to architecture

Table 4: Systems objects used in the experiment.

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

recovery. Figure 3(a) and Figure 3(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.

4.2 Execution

In order to level the knowledge of the participants, basic architecture concepts were introduced for each participant, 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 [17] ,which is based on visualization via dependency graphs and dependency structure matrix. While to analyse the Java systems, the subjects used two different tools: `X-Ray` [10] and `Architexa` [15]. Both Java analysis tools are plug-ins for the Eclipse IDE. Through these tools, one can get different views of class-package dependencies and systemic complexity views on Java projects. The `Bunch` tool [12] was used for the automatic approach of the architecture recovery. Participants were instructed to use the default Nearest Ascent Hill-Climbing algorithm (NAHC) implemented in the tool.

For each of the two participating groups, we randomly selected one system in Java and one system in Visual Basic, so that in the end each group analyses systems in both languages. Then, each participant started the architecture recovery procedure by applying two reengineering approaches per session, which produced two models representing the modular decomposition of the system. In the first part of the 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. We also made sure that the application order of each technique was prioritized in one of the two systems in each group. In this way, it was possible to calculate the accuracy of the resulting models using one technique, two techniques while varying their order of application.[1].

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 the participants of the experiment.

[1]The resulting models are available at https://github.com/ArchRecovery-SAC2016/files

4.3 Results

To analyze the significance of the results, it was used the paired `t-test`. To use such parametric approach, 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.

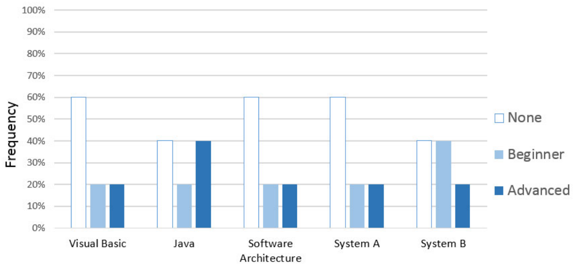
	Technique	N	Means	Std. Deviation	Std.Error Mean
Visual Basic	One Technique	10	0.517	0.091	0.028
	Two Techniques	10	0.771	0.077	0.024
Java	One Technique	10	0.483	0.060	0.019
	Two Techniques	10	0.728	0.059	0.018

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`.

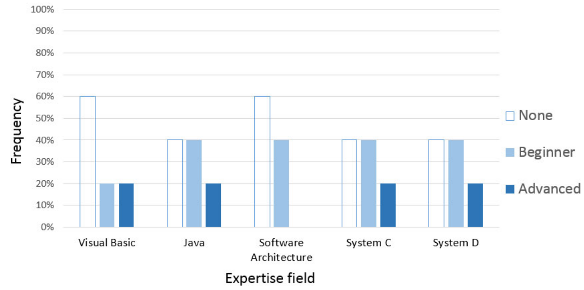
Table 6: Paired t-test for Visual Basic systems.

	Paired Diferences							
				95% Confidence Interval				
	Mean	Std. Deviation	Std. Error Mean	Lower	Upper	t	df	Sig. (2-tailed)
(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

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 ($Mean = 0.483$, $Std.\ Deviation$ =



(a) Participants expertise in Group 1.



(b) Participants expertise in Group 2.

Figure 3: Expertise of the participants.

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.

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 significantly impacts the accuracy of the models produced, since all the results indicated an accuracy increase 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%. Using first clustering followed by 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
Visualization-Clustering	10	0.771	0.077	0.024
Clustering-Visualization	10	0.728	0.059	0.018

To investigate the statistical significance of the difference between the means, it was conducted a paired `t-test`. The results are shown in Table 8. Results show that, with 95% confidence interval, the lower and upper limit range includes 0. As a result, it is possible to conclude that there is no statistical significance of the difference between the mean values obtained using software visualization prior to clustering or vice versa.

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

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

4.4 Discussion

Altogether, the results showed that the accuracy of the models produced had a significant increase when two techniques were used together compared to using one technique alone, irrespective of the technique order applied. 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 techniques execution order, there was no common agreement among the participants in both groups. Thus, our results indicate that the achievement of accuracy does not require a particular execution order of the techniques, which might happen in a subjective way that best meets the needs of the responsible for the architecture recovery.

4.5 Threats to validity

Regarding internal validity, our study is limited to the choices we made with respect to the analysis tools used in the experiment and the object systems used in the analyses. However, we should note that the selected analysis tools have been widely used either in industry or in academia.

Some limitations to our results might also arise due to the the participants selection. In order to avoid biased results, the groups participants were randomly selected and both groups have comparable experience in analysis and development of software systems in VB and Java. Despite the fact that the participants were separated into two groups and ran the experiment in different objects systems and programming languages, the recovery approach was conducted in systems with similar architecture styles. Due to this similarity, it is expected that the results were not influenced by the programming language objects, or difference between the systems, as we make this evident via statistical testing.

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.

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. D. Penta, and L. Cerulo. Achievements and challenges in software reverse engineering. *Commun. ACM*, 54(4):142–151, 2011.

[2] S. Ducasse and D. Pollet. Software architecture reconstruction: A process-oriented taxonomy. *IEEE Trans. Software Eng.*, 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] 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] T. Lutellier, D. Chollak, J. Garcia, L. Tan, D. Rayside, N. Medvidović, and R. Kroeger. Comparing software architecture recovery techniques using accurate dependencies. In *Proceedings of the 37th International Conference on Software Engineering - Volume 2*, ICSE '15, pages 69–78, Piscataway, NJ, USA, 2015. IEEE Press.

[10] J. Malnati. X-Ray web site. http://xray.inf.usi.ch/xray.php, 2015. [Online; accessed 20-september-2015].

[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 approach to solving the software clustering problem. In *19th International Conference on Software Maintenance (ICSM*, pages 285–288, 2003.

[13] M. C. Platenius, M. von Detten, and S. Becker. Archimetrix: Improved software architecture recovery in the presence of design deficiencies. In *16th European Conference on Software Maintenance and Reengineering, CSMR 2012, Szeged, Hungary, March 27-30, 2012*, pages 255–264, 2012.

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

[15] V. Sinha and C. Deschenes. Architexa web site. http://www.architexa.com/, 2015. [Online; accessed 20-september-2015].

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

[17] VBDepdent-CoderGears. VBDepend web site. http://www.vbdepend.com/, 2015. [Online; accessed 20-september-2015].

[18] 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.

[19] 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.