A Survey Paper on Software Architecture Visualization

Yaser Ghanam and Sheelagh Carpendale
Department of Computer Science
University of Calgary, Canada T2N 1N4
{yghanam, sheelagh}@ucalgary.ca

ABSTRACT

Understanding the software architecture is a vital step towards building and maintaining software systems. But software architecture is an intangible conceptual entity. Therefore, it is hard to comprehend a software architecture without a visual mapping that reduces the burden on the human brain. Visualizing software architecture has been one of the most important topics in software visualization. Not only are architects interested in this visualization but also developers, testers, project managers and even customers. This paper is a survey on recent and key literature on software architecture visualization. It touches on efforts that defined what characteristics an effective visualization should have. It compares various efforts in this discipline according to taxonomies such as dimensionality, multiplicity of views and use of metaphors. The paper also discusses trends and patterns in recent research and addresses research questions that are still open for further investigation.

Author Keywords
Software architecture visualization.

ACM Classification Keywords
Software, visualization, HCI.

1. INTRODUCTION

1.1. Software Visualization

Simply put, Software Visualization (SV) is the use of visual representations to enhance the understanding and comprehension of the different aspects of a software system. Price et al. [1] gives a more precise definition of software visualization as the combination of utilizing graphic design and animation combined with technologies in human-computer interaction to reach the ultimate goal of enhancing both the understanding of software systems as well as the effective use of these systems. The need to visualize software systems evolved from the fact that such systems are not as tangible and visible as physical objects in the real world [2]. This need becomes particularly evident when the software system grows to entail a huge number of complexly related modules and procedures. This growth results in a boost in the time and effort needed to understand the system, maintain its components, extend its functionality, debug it and write tests for it.

1.2. Scope

SV is a broad field that is concerned with visualizing aspects of software engineering as a practice and software systems as evolving products. Some of these aspects include design models and patterns, software architecture, development processes, code history, database schemes, network interactions, web services, parallel processing, process execution and many others [3]. This paper focuses on a single aspect of SV which is software architecture. Software architecture refers to the structure of a software system including composing entities, the metrics of these entities, and the relationships among different entities [4]. Visualizing software architecture encompasses not only the software modules and their internal structures and interrelations, but also the evolution of these modules over time [5]. Considering the numerous tools and approaches proposed to directly or indirectly help build an understanding of the system architecture (including software exploration tools), research in this area has been extensive especially in the last few years.

1.3. Structure

Section 2 of this paper is a listing of the problems and questions researchers in the field of software architecture visualization have been attempting to address and solve. Then, Section 3 touches on the criteria of effective software visualizations. Section 4 follows to discuss the tools and techniques developed to design and build effective architecture visualizations. Section 5 talks about the main trends in the techniques and tools used to produce architecture visualizations followed by a discussion on open research questions in this discipline.

2. QUESTIONS & PROBLEMS

In this section, we touch on the main problems and questions recent research has been trying to tackle.

Research in the area of software architecture visualization is centered on finding a meaningful and effective mapping scheme between the software architecture elements and visual metaphors [6]. Recent research has been trying to answer different questions such as: “why is the

visualization needed?", "who will use [it]?", and "how to represent it?" [7]. Others like [8] questioned the effectiveness and expressiveness of the visuals to use. In general the various questions asked in this discipline can be grouped into three broad categories:

- Who are the different groups of audience for architecture visualization [9]?
- What questions do they wish to answer through this visualization [10]?
- How can visual metaphors and interaction techniques be used to answer their questions [11]?

As will be discussed later in this paper, these three questions can be thought of as determinants of what is to be visualized and how.

2.1. Who?

By answering the "who" question, we determine the different groups of audience for architecture visualization and their various interests.

Defining the audience of the architecture visualization plays a pivotal role in determining what to visualize and how to visualize it. McNair et al. [9] defines three different groups that are interested in the software architecture of a given system. These three groups are developers, managers and researchers. Panas et al. [10] consider architects to be one additional group, and they differentiate between developers and maintainers.

2.2. What?

Having defined the audience for architecture visualization, it is equally important to understand the needs of this audience. These needs can be understood by studying the roles different groups of audience play and the influences they have on the software architecture.

The questions these groups address differ according to their various interests and level of involvement in the software project. Therefore, the interests of these groups are to be considered when visualizing software architecture. For example, it is important for architects to realize the different characteristics of the architecture they design such as complexity, coupling, cohesion and other attributes. Developers (and maintainers) are usually interested in understanding how the architecture is laid out, and what the most recent status of the software system is. Developers deem this understanding important because it enables them to maintain the system and continue the development process. Managers; on the other hand, want, on a high level, to be able to monitor the progress of the project and determine the completion of development goals. On another level are researchers. They are interested in studying characteristics and trends of software architectures and their evolution in general. Customers are also considered an important audience for architecture visualization [12]. They might ask or need to have a general overview about the design of the system, but they need not look at low level details that are irrelevant to their concerns. Figure 1 illustrates the previously mentioned groups along with their interests in the various levels of detail in the software architecture based on [9], [10] and [12].

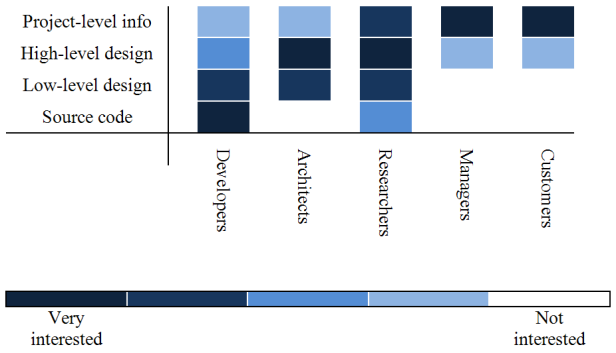

Figure 1 - Various levels of interest of SV audience

2.3. How?

To satisfy the needs of the different stakeholders, the visualization needs to be capable of answering their various questions about the software architecture. Numerous approaches, techniques and tools have been proposed by researchers in an attempt to produce an effective visualization that can achieve this goal. But what an effective visualization is can be an ambiguous issue that inevitably might create a gap between the proposed solutions and the needed ones. This issue of determining the effectiveness of the visualization has driven a new stream of research dedicated to answering the question: "What makes architecture visualization effective?" The following section will discuss some of the efforts to establish reliable and valid criteria that determine the effectiveness of a proposed solution.

3. CRITERIA FOR AN EFFECTIVE VISUALIZATION

Research in software architecture visualization attempts to answer the spacious variety of questions asked by the different stakeholders through a number of techniques and approaches. However, what determines how effective a specific visualization technique is has never been a trivial question. Storey et al. [13] conducted a comprehensive analysis of cognitive models of program comprehension. The aim of her work was to find a set of criteria against which the effectiveness of software exploration tools can be measured. In this highly referenced paper [13], she presented a hierarchy of cognitive issues which should be considered when developing a software exploration tool as partially shown in Figure 2.

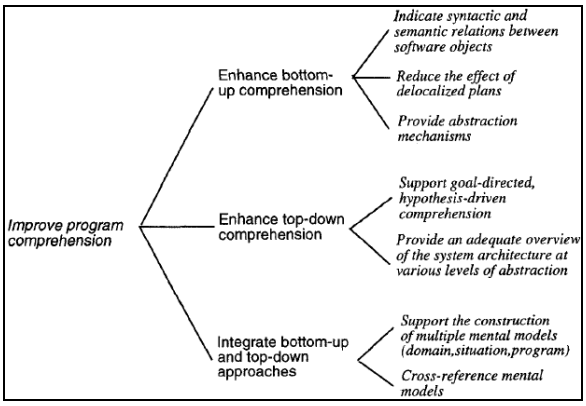

Figure 2 - A part of the hierarchy of cognitive issues defined by Storey et al. [13]

Amongst the different cognitive models mentioned in this paper are two fundamental models. The first model is the bottom-up comprehension model, where software understanding is constructed from mentally grouping smaller elements into higher level abstractions. The top-down comprehension model; however, states that software understanding is constructed through a high level comprehension of the domain of the software followed by a mapping of this domain into smaller building components. Some research suggests that people are actually capable of using either model according to certain cues. Whereas an integrated model suggests that people can simultaneously activate both models at any given time to gain an understanding of the software at different levels of abstraction. The implications of the work in [13] and other efforts in the same field by Storey such as [14] and [15], have been influential on recent and current research [6, 10, 16, 17, 18]. When designing a software visualization tool, Storey states that it is essential to determine whether the tool better support a top-down approach, a bottom-up approach or a hybrid of both.

Li et al. [19] had another interesting view on what defines an effective visualization based on the challenge of complexity in software architectures. According to [19], in addition to the data complexity represented in the huge number of nodes and edges, there is a visual complexity associated with the limitations of the human brain capabilities and short-term memory capacity. Therefore, Li et al. proposed the use of a clustered graph model to reduce the data complexity, and, consequently, the visual complexity. In this model, three requirements are to be satisfied in order to deem the view of the architecture effective: an adequate information supply (a non-distorting level of abstraction), mental map requirements (the context of previous views should be preserved), and a constant visual complexity. Also, the algorithm for laying out the visual elements should ensure a non-overlapping distribution of nodes, the ability to produce compact output and a way to preserve the mental map. Young et al. [20] also suggested that low visual complexity is a desirable property in any visualization. They also added other criteria to measure against when evaluating the effectiveness of a specific visualization such as simple navigation, good use of visual metaphors and good use of interaction.

4. SOLUTIONS: TOOLS & TECHNIQUES

Efforts in the field of architecture visualization can be categorized in more than one way. The following is a comparative review of existing solutions based on three taxonomies: multiplicity of view, dimensionality and metaphor. In the following survey, these characteristics will overlap resulting in some papers being mentioned more than once under different sections.

4.1. Multiplicity of view

With regards to the multiplicity of view, two schools of thoughts can be identified. On the one hand, the first school asserts that any visualization should support multiple views of the architecture at different levels of detail in order to satisfy the audience's different interests [18, 21, 22]. That is, for the visualization to be deemed useful, it has to provide a means of looking at the different aspects of a software architecture through different views, and possibly via multiple windows. For instance, if one view provides an insight into the internal structure of software entities composing the architecture, another view should, on a higher level, focus on the relationships and communication between these entities.

The other school of thought; on the other hand, believes that a carefully designed single view of the visualization might be more effective and meaningful in conveying the multiple aspects of the architecture than the multiple view approach [10, 14]. For example, the tool can provide different levels of detail in a single view (e.g. internal structures of entities along with the relationships between them) and leave it up to the viewers to draw their own mental maps at the level of interest to them.

4.1.1. Multiple-view visualization

Lanza et al. [23, 24] introduced a software visualization tool called CodeCrawler (CC). Through a 2D visualization of reverse-engineered object oriented software systems, CC offers the advantage of having multiple views of the same architecture. The multiplicity of views aims to uncover the different aspects of the architectural design and emphasize specific metrics in the software system. In general, CC represents the architecture in a polymeric view in which entities (classes, methods … etc) are represented as nodes and relationships as edges connecting these nodes. The node's size, position and color are used to represent the metrics of interest in the software system. There are four different views that CC has to offer: 1) coarse-grained view by

which the system complexity is emphasized, 2) fine-grained view which hierarchically represents the class blueprint in the architecture, 3) coupling view which, as the name suggests, underscores coupling amongst modules in the architecture, and 4) evolutionary view which helps track changes of the architecture over time. Figure 3 shows the four different views. Lanza et al. explain, in great detail, the algorithm followed to layout the class blueprint visualization in a separate paper [25].

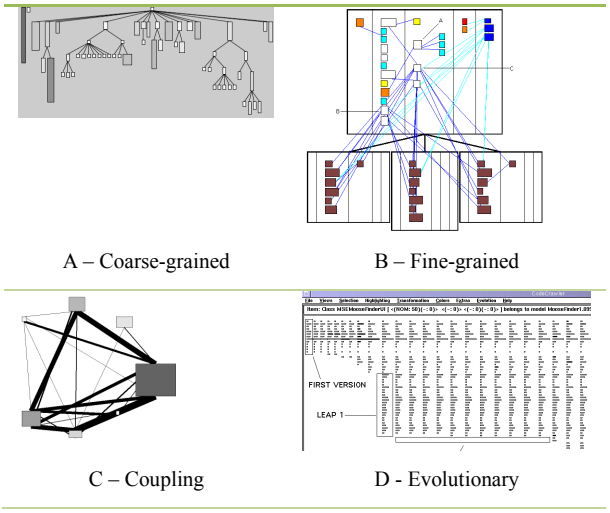

Figure 3 - Four different views for the software architecture by Lanza et al. [23, 24]

4.1.2. Single-view visualization

SHriMP [14, 15] was one of the early attempts to visualize multiple aspects of the architecture through a single view. Storey et al. [14] suggested the use of a unified single view visualization that presents information at different levels of the software system, especially the architectural level. However, according to Panas et al. [10], SHriMP did not consider stakeholder communication. Therefore, [10] proposed an enhanced single-view model that addresses three main issues. For one, it enhances communication between the different stakeholders by allowing them to reach a common understanding of the architecture. Secondly, it reduces the "significant cognitive burden" resulting from trying to comprehend multiple views. And thirdly, it rapidly summarizes systems especially large-scale ones. The proposed model uses common (rather than varying) interests amongst stakeholders to come up with a collective comprehension of the architecture. Figure 4 shows an example of this visualization.

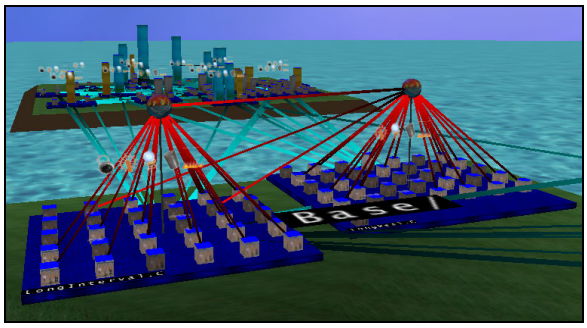

Figure 4 - A unified view of software architecture by Panas et al. [10]

In the figure above, the green landscapes represent directories (high level groups); whereas the blue plates indicate source files. And methods are represented by buildings.

4.2. Dimensionality

Visualizations in general can be classified according to the dimensionality of the graphical elements as 2D visualizations and 3D visualizations. Virtual Environments (VE) are also used as a means of conveying information about software architectures especially large-scale ones. VEs can mainly be categorized under 3D visualizations. This paper; however, discusses VEs as a separate group of visualizations due to the considerable number of focused research efforts on this specific technique.

4.2.1 2D visualization

By differentiating between 2D and 3D representations, we refer to the dimensionality of the graphical visuals and not the dimensionality of the data itself. That is, a carefully designed 2D representation of an architecture should be capable of representing more than two dimensions in the dataset. For example, if we look back at Figure 3, we can notice that CodeCrawler [23, 24] chose to use a two dimensional representation of the software architecture. That is, there is no third dimension in the graphical visuals such as depth. Nevertheless, CC used the attributes of the 2D visuals to reflect more dimensions. The node size, color and position are all used to visualize more characteristics of the software entities. Figure 5 shows the metrics in the 2D visual that were utilized to map the entity's attributes.

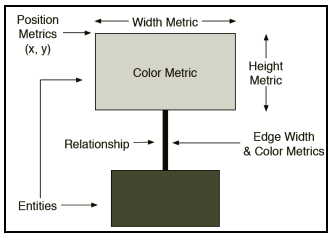

Figure 5 - The metrics used in the 2D visual to map the entity's attributes by Lanza et al. [23, 24]

Furthermore, Storey et al. [14, 15] suggested the use of the SHriMP (Simple Hierarchical Multi-Perspective) tool as a customizable and interactive 2D environment for navigating and browsing hierarchical software structures. The tool, originally described in [26], uses filtering, abstraction and graph layout algorithms to reveal complex structures in the software system. The hierarchical structure is represented via a nested graph in a containment layout as shown in Figure 6.

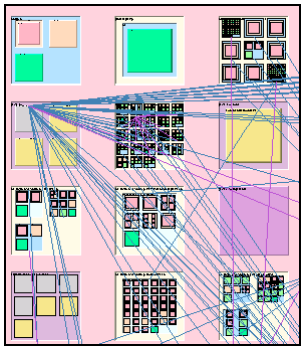

Figure 6 –SHriMP showing the extend & implement relationships between classes in a software architecture by Storey et al. [14, 15]

The tool provides fisheye zooming, semantic zooming as well as geometric zooming to serve different purposes of navigation. Also, the capability to hide some levels of the architecture is provided in order to make the visualization suitable for different levels of details. Although this work was a relatively simplistic design, it has been repeatedly referenced in literature and was the basis for later efforts in architecture visualization [11, 16, 27, 28].

4.2.2 3D visualization

The main reason behind proposing the use of 3D visualizations, as will be detailed in this section, is the realization that using only two dimensions to represent highly dimensional data can be too overwhelming for the viewer to comprehend. In an early attempt to justify the use of the third dimension, [29] mentions that visualizing a 1000 node cone tree without visual clutter can in no way be achieved in a 2D plane, but is possible using a 3D plane.

Marcus et al. [16] suggested the utilization of the third dimension to better represent highly dimensional data in large-scale software systems. This 3D visualization, as shown in Figure 7, does not use graph-based representation because, according to [16], graph-based representations suffer from scalability, layout and mapping problems.

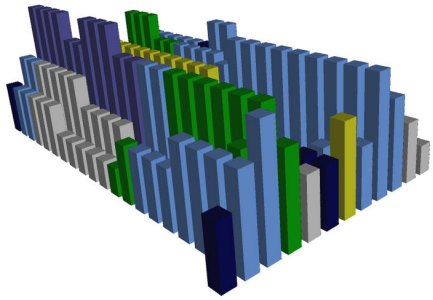

Figure 7 - A non-graph-based representation by Marcus et al. [16]

But rather it uses bar-based color and pixel maps to show relationships between elements of a software system. The reason behind introducing the third dimension is that 2D pixel bars limit the number of attributes that can be visualized. Occlusion was a problem in this visualization until transparency and elevation control was introduced as shown in Figure 8.

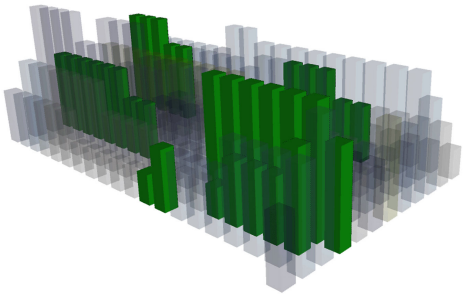

Figure 8 - Transparency as a solution for occlusion by Marcus et al. [16]

It is worthwhile to mention that there was a previous effort to use brightness to represent the third dimension, but it did not enjoy success because it was confusing and poorly perceived by the users [36].

DiffArchViz is another tool developed by Sawant et al. [18] specifically for visualizing the software architecture of network-based large-scale systems. According to [18], the use of 3D glyphs to represent the architecture was to offer more surface area to place information on. This tool uses a multidimensional scaling technique to represent the pattern of proximities among the set of software components extracted from raw code. The visualization uses hue, luminance, size, orientation, transparency and height as visual representations for the different attributes

of software components (represented as glyphs). Figure 9 shows a screenshot of this visualization. The drawback of this tool is that it categorizes raw code into software components in a static manner, hence mandating the user to stick to this categorization throughout the visualization process.

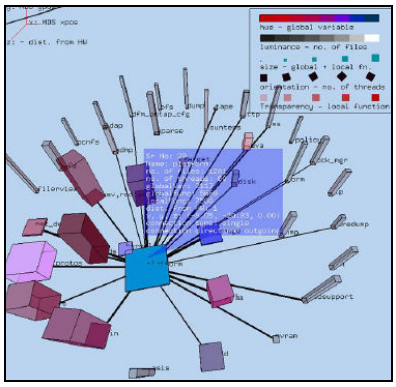

Figure 9 - Glyphs as a 3D visualization metaphor by Sawant et al. [18]

4.2.3 VE visualization

The ultimate goal of using Virtual Environments to visualize software architecture is to make it possible and easy for the viewer to compare metrics of the different components in the architecture and realize the relationships amongst them. Also, amplifying cognition is another advantage of VEs, for they allow for navigation in an open 3D space that has commonalities with physical environments in our everyday life.

EvoSpaces [11] is a reverse engineering tool that provides an architectural level visualization of software systems as a virtual environment. It takes advantage of the fact that software systems are often structured hierarchically to suggest the use of a virtual city metaphor. Entities along with their relationships are represented as residential glyphs (e.g. house, apartment, office, hall … etc); whereas metrics of these entities are represented as positions and visual scales in the 3D layout (e.g. size, color value … etc). The tool provides different interaction modes with zooming and navigation capabilities. Figure 10 shows an instance of visualizing a software architecture using EvoSpaces.

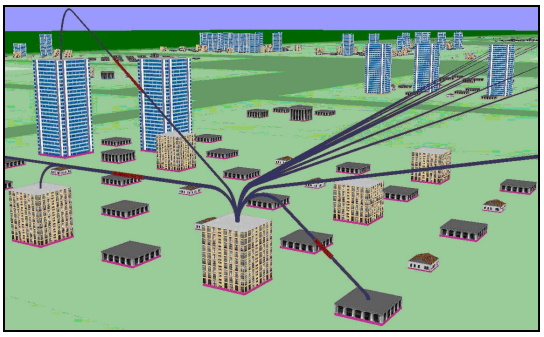

Figure 10 - EvoSpaces virtual city by Alam et al. [11]

Users can navigate through this virtual city with a road map that would be available in one of the corners of the screen. They can zoom inside buildings to see stickmen who represent methods and functions. Each stickman may be surrounded by his resources (yellow-colored boxes) which represent local variables.

In addition, Panas et al. [10], as we showed earlier in Figure 4, also used the city metaphor as a theme for a virtual environment. This work aimed for a reduction of the visual complexity of the single-view visualization Panas proposed.

4.3 Metaphor

As discussed earlier in this paper, a primary aspect of research in the area of architecture visualization is finding a meaningful mapping scheme between the software architecture elements and visual metaphors [6]. Some researchers suggested the use of abstract metaphors such as polygons, spheres and other geometrical 2D or 3D shapes to represent software entities. Another group of researchers thought it would be easier for the human brain to understand the structure of and the relationships in a software architecture if a real metaphor was used instead. That is, it is argued that by looking at familiar objects (such as buildings) whose physical characteristics map directly to attributes of the software entities, the viewer will perceive the visualization as more intuitive and easier to comprehend and remember. On the negative side; however, the complexity of the real metaphor may negatively impact the effectiveness of the visualization as will be discussed later.

4.3.1. Abstract metaphor

Balzer et al. suggested the use of 3D spheres [31] or hemispheres [32] as an abstract metaphor to visualize entities in software architecture (e.g. packages, classes, methods and attributes). These entities are related to each other through inheritance, access and call relationships. Nested hemispheres are used in a containment layout to represent the hierarchical structure of a software system with their size corresponding to how many entities they contain. Moreover, relationships are visualized using a

"Hierarchical Net" where a point resting above the center of each hemisphere is used as a connecting hub for the relationships within this hemisphere. The thickness of a connecting edge between two nodes is used as an indicator of the number of relations that link these nodes. An interesting use of transparency in the suggested solution makes it possible to view into the system or sub-system while maintaining the context of neighboring sub-systems. The deeper the entity is in the software hierarchy, the more opaque it becomes. Figure 11 shows an example of the hemisphere visualization.

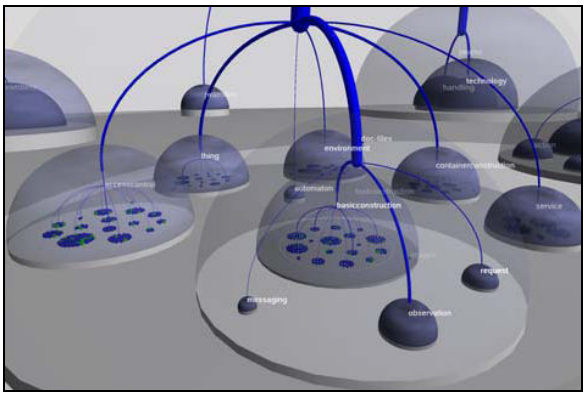

Figure 11 - Hemispheres as an abstract metaphor by Balzer et al. [32]

A year later, Balzer et al. [33] proposed another solution to visualize metrics of a software architecture using Treemaps. It is also a hierarchy-based solution with an abstract metaphor. But the main difference in this effort is the use of the polygon metaphor to visualize entities.

4.3.2. Real metaphor

The main goal that drives some efforts to adopt the use of real metaphors in architecture visualization is to utilize the properties of familiar objects (such as houses and tables) to enhance the users' understanding of software components. For instance, Boccuzzo [12] introduced CocoViz as a software metrics configuring engine to handle metaphors and allow optimizations to their graphical representation. Depending on the fact that it is easy for a viewer to quickly distinguish between a well-shaped glyph and a miss-shaped one, the distinction between a good architectural design and a poor one will be fairly easy and quick. CocoViz uses three different metaphors to represent entities in a software architecture: house metaphor, table metaphor and spear metaphor. For each metaphor, a set of parameters is defined along with a criterion that determines whether a well-shaped glyph or a miss-shaped glyph should be used for representing a given entity as illustrated in Figure 12.

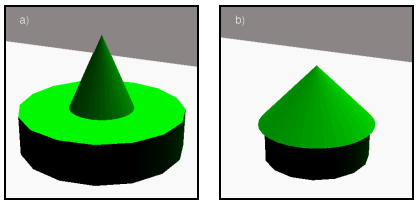

Figure 12 - A house metaphor showing a) a miss-shaped and b) a well-shaped glyph by Boccuzzo [12].

The tool also provides normalization and filtering capabilities to make the visualization more cognitively intuitive. Figure 13 shows the mapped software metrics in the house metaphor representation.

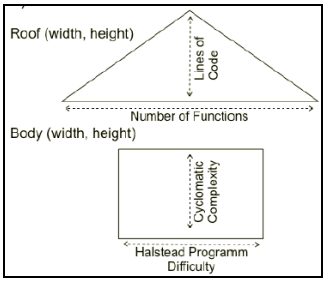

Figure 13 - The mapped software metrics in the house metaphor by Boccuzzo [12].

Beside the works previously shown in Figure 4 and Figure 10 where the city metaphor was used, Vizz3D [35] is another tool that used the very same metaphor and later was the basis for other efforts like the one by [10]. Figure 14 shows the abstract representation of a software architecture along with the city metaphor mapping of that architecture.

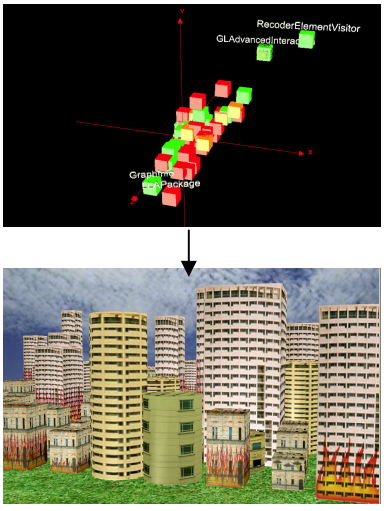

Figure 14 - Mapping between abstract & real metaphor in Vizz3D [35]

At different levels of complexity, the house metaphor seems to repeatedly be used in architecture visualization as a real metaphor. This metaphor is hoped to make the software entities and the relationship between them cognitively easy to comprehend [12, 34]. A collection of house-related metaphors are utilized to come up with a city metaphor that is more complex in nature and potentially more capable of visualizing the different perspectives of a given software architecture [10, 11, 34].

5. DISCUSSION

5.1. Trends

In this paper, the main focus of our literature review was research efforts in the last few years. It is indeed difficult to make conclusions about trends in the discipline of software architecture visualization without also looking at research efforts in the last two decades. This is because the incremental advancements in this field, as you might have noticed from the previous sections, are not significant enough to notice revolutionary changes in a short period of time. Actually, some works are too similar to be able to tell what the significance of the latter work is. For example, in 2007, the "habitability" or the "city" metaphor has been a repeatedly proposed visualization metaphor with very small incremental enhancements [10, 11, 35]. For these reasons, the reader is asked to keep in mind, when reading the analysis in this section, that little context has been considered before the year 2001.

The plot in Figure 15 shows some of the papers in the architecture visualization literature (some of which have been discussed in detail in Section 4) over a 7 year period. The x-axis represents the years whereas the y-axis indicates the references to these works in our paper. A box can be either white (2D), grey (3D) or black (VE). A dotted frame indicates multiple-view whereas a connected frame indicates single-view. Each box contains either the letter "A" for abstract metaphor or "R" for real metaphor.

As seen in Figure 15, the most obvious trend in software architecture visualization (SAV) is the use of real metaphors as opposed to abstract ones. By 2005, most of the proposed efforts focused on delivering visualizations that make use of real-life objects to represent software entities. This trend has been most probably supported by the advancement in graphics-related technologies (software and hardware) rather than empirical evidence of the advantages of using a real metaphor in software visualizations. Literature lacks empirical evaluation of what the added benefit is when utilizing a "city" metaphor for example. There have been evaluations that tested utilization of space and other visual attributes, but none (from the set covered) touched on the cognitive aspect of the effectiveness of these real metaphors.

Adding an extra dimension for existing 2D visualization solutions has also been a noticeable trend in recent literature. 3D visualizations became more appealing after the advantages of 3D visualizations over 2D ones were uncovered through practical comparisons [16]. Although in 2005, there were still some proposed 2D solutions, we can notice that in 2007, most (if not all) of the proposed solutions consider the third dimension. This transition to 3D spaces can be deemed an enhancement (as opposed to the previous trend of real metaphors) because of the accompanying (old and new) evidences of what the added benefits are.

Considering the third taxonomy, there is no obvious trend other than the original practice of providing multiple views of the architecture to underscore the different design aspects. Some people did suggest that single-view visualizations are more effective, but their argument was not backed by any experiments or empirical studies.

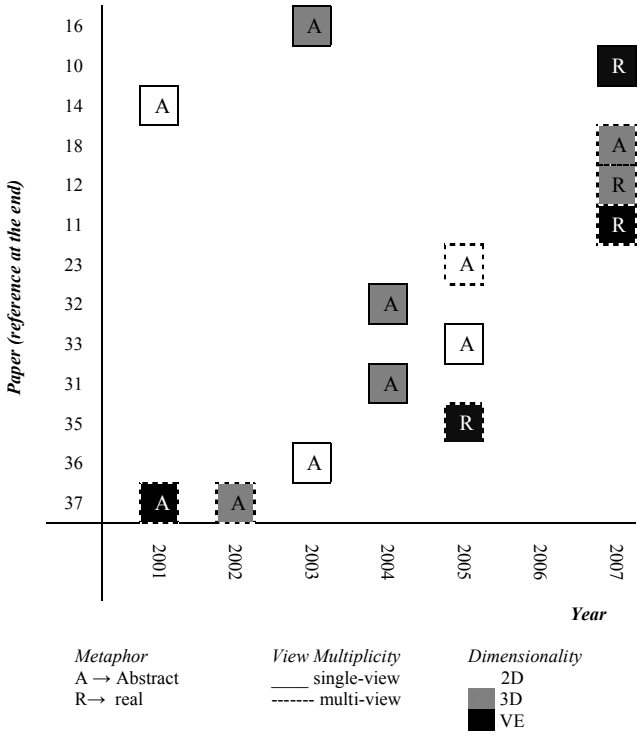

Figure 15 - Some papers on SAV from 2001 to 2007

5.2. Open research questions

In Section 2 of this paper, we mentioned that the questions research has been trying to answer can be summarized by: Who is the audience? What do they want to know? How can the visualization satisfy their needs? And we also mentioned some efforts to identify criteria for good visualizations.

It is fairly reasonable to assume that research efforts have successfully identified the different groups of audience and their various interests. This conclusion can be drawn

from the fact that recent papers have (in most of the cases) designed and structured their work to target a well-defined audience.

Having said that, it is important to mention that these efforts have failed in evaluating how their work directly influenced the targeted audience. Some people did evaluate their work against some criteria and heuristics. But as mentioned before, there has not been enough evaluation that considers human factors in architecture visualization. Therefore, it is not guaranteed that the tools and techniques being developed satisfy the audiences' needs even though they might fully satisfy design principles.

Moreover, it is not sufficient to rely on our guesses or use the "coolness" factor to decide whether a specific metaphor should be used or not. There is a need to study how effective and expressive an abstract or a real metaphor is. This is still an open question that requires a huge amount of effort to answer. Not only do we need to address the cognitive influence of these metaphors on the viewer's comprehension of the architecture, but we also need to study the correlation between the complexity of the metaphor and the viewer's ability to form the intended mental map. This issue was also raised by Knodel et al. [38] who confirmed that there are no empirical evidences of the benefits of some visualization concepts in software architecture. Therefore, [38] introduced an approach that explicitly integrated architecture development with the selection, use and empirical validation of visualization metaphors. The approach is a composition of defined roles (i.e. stakeholders & visualization experts), processing steps (i.e. selection of architectural perspectives), and products (i.e. architectural view). This approach was validated through several industrial projects and proved to be successful.

6. CONCLUSION

Software visualization is a broad topic that has increasingly been extended to cover many aspects of software engineering and software systems. Two main conferences lead research in this area: "The ACM Symposium on Software Visualization (SOFTVIS)" and "The IEEE International Workshop on Visualizing Software for Understanding and Analysis (VISSOFT)". Other conferences like SIGGRAPH and INFOVIS are also valuable venues for this discipline.

This paper talked about a specific area in software visualization concerned with visualizing software architecture. To begin with, we defined the field and its position within the scope of software visualization. Then, we discussed the three main axes around which all efforts in this field orbit, namely: the audience, their needs and the tools and techniques to satisfy these needs. After that, we mentioned the attributes against which, as researchers suggested, visualization efforts should be evaluated such as expressiveness, user interaction, visual complexity and navigation techniques.

The fourth part of this paper focused on recent software architecture visualization tools and techniques. We defined three taxonomies to differentiate between the various efforts. The first taxonomy is dimensionality. We noticed that there is an obvious and justifiable trend to make use of the third dimension as a way to reduce the visual complexity. In the second taxonomy, multiplicity of views, we noticed no specific trends but different opinions about whether a single-view or a multiple-view is more effective in representing the multiple aspects of the software architecture. The most obvious trend in literature was related to the third taxonomy which is metaphors. There is an increasing tendency to utilize real metaphors as opposed to abstract ones as a means to amplify cognition. But with no empirical evidence of the added benefits of these real metaphors, this tendency is yet to be scientifically justified.

Open research questions in the field of software architecture visualization are mainly related to the validation of the effectiveness of the tools and approaches proposed. There is an evident lack of empirical evidence which is an important step towards understanding the value of the research done in this field.

This survey paper was limited to recent and key research efforts. A more longitudinal literature survey is needed in order to come up with more valid and reliable generalizations about trends and patterns in software architecture visualization research.

ACKNOWLEDGMENTS

We thank Dr. Carpendale for reviewing the credibility of the resources used in this literature review and making sure the paper collection sufficiently covers the topic of interest.

REFERENCES

1. Price, B.A., Baecker, R., and Small, I.S. (1998) A principled taxonomy of software visualisation. *Journal of Visual Languages and Computing*. 4(3), pp. 211-266.
2. Petre, M., and de Quincey, E. (2006) A gentle overview of software visualization. *The Computer Society of India Communications (CSIC) & Autumn 2006 PPIG newsletter*.
3. ACM Symposium on Software Visualization. Available at http://www.st.uni-trier.de/~diehl/softvis/org/softvis08, last accessed Feb 20, 2008.

4. L. Bass, P. Clements, and R. Kazman. (2003) *Software Architecture in Practice*. Addison – Wesley Inc.
5. D'Ambros, M., and Lanza, M. (2007) BugCrawler: Visualizing Evolving Software Systems. *The 11th European Conference on Software Maintenance and Reengineering, 2007*, pp.333-334.
6. Gračanin, D., Matković, K., and Eltoweissy, M. (2005) Software visualization. *Innovations in Systems and Software Engineering*, 1(2), pp. 221-230, Springer London.
7. Maletic, J., Marcus, A., and Coollard, M. (2002). A Task Oriented View of Software Visualization. *IEEE Workshop of Visualizing Software for Understanding and Analysis, 2002*, pp. 32-40.
8. Mackinlay, J. (1986) Automating the design of graphical presentation of relational information. ACM Transaction on Graphics, 5(2), pp. 110-141.
9. McNair, A., German, D., Weber-Jahnke, J. (2007) Visualizing Software Architecture Evolution Using Change-Sets. *The 14th Working Conference on Reverse Engineering, 2007*, pp.130-139.
10. Panas, T., Epperly, T., Quinlan, D., Saebjornsen, A., and Vuduc, R. (2007) Communicating Software Architecture using a Unified Single-View Visualization. *The 12th IEEE International Conference on Engineering Complex Computer Systems, 2007*, pp. 217-228.
11. Alam S., and Dugerdil P. (2007) EvoSpaces: 3D Visualization of Software Architecture. *The 19th International Conference on Software Engineering & Knowledge Engineering, 2007.*
12. Boccuzzo, S., and Gall, H. (2007) CocoViz: Towards Cognitive Software Visualizations. *The 4th IEEE International Workshop on Visualizing Software for Understanding and Analysis, 2007*, pp.72-79.
13. Storey, M., Fracchia, F., and Muller, H. (1997) Cognitive design elements to support the construction of a mental model during software visualization. *The Fifth International Workshop on Program Comprehension, 1997*, pp.17-28.
14. Storey, M., Best, C., and Michand, J. (2001) SHriMP views: an interactive environment for exploring Java programs. *The 9th International Workshop on Program Comprehension, 2001*, pp.111-112.
15. Storey, M., Best, C., Michaud, J., Rayside, D., Litoiu, M., and Musen, M. (2002) SHriMP views: an interactive environment for information visualization and navigation. In *CHI '02 Extended Abstracts on Human Factors in Computing Systems, 2002.*
16. Marcus, A., Feng, L., and Maletic, J. (2003) 3D Representations for Software Visualization. *The 1st ACM Symposium on Software Visualization, 2003*, pp. 27-36.
17. Tilley, S. and Huang, S. (2002) Documenting software systems with views III: towards a task-oriented classification of program visualization techniques. *The 20th Annual international Conference on Computer Documentation*, 2002, pp. 226-233.
18. Sawant, A., and Bali, N. (2007) DiffArchViz: A Tool to Visualize Correspondence between Multiple Representations of Software Architecture. *The 4th IEEE International Workshop on Visualizing Software for Understanding and Analysis, 2007*, pp.121-128.
19. Li, W., Eades, P., and Hong, S. (2005) Navigating software architectures with constant visual complexity. IEEE Symposium on Visual Languages and Human-Centric Computing, 2005, pp. 225-232.
20. Young, P., and Munro, M. (2003) Visualising software in virtual reality. *IEEE First International Workshop on Visualizing Software for Understanding and Analysis, 2003*.
21. Reiss, S., and Renieris, M. (2003) The BLOOM Software Visualization System. *Software Visualization – From Theory to Practice*, MIT Press, 2003.
22. Kruchten, P. (1995) The "4+1" view model of architecture. *IEEE Software*, 12(6), pp. 42-50.
23. Lanza, M., Ducasse, S., Gall, H., and Pinzger, M. (2005) CodeCrawler - an information visualization tool for program comprehension. *The 27th International Conference on Software Engineering, 2005*, pp. 672-673.
24. Lanza, M. (2003) CodeCrawler - A Lightweight Software Visualization Tool. *The 2nd International Workshop on Visualizing Software for Understanding and Analysis, 2003*, pp. 51 - 52.
25. Lanza, M., and Ducasse, S. (2001) The Class Blueprint - A Visualization of the Internal Structure of Classes. *Software Visualization Workshop (OOPSLA, 2001).*
26. Michaud, J., Storey, M., and Muller, H. (2001) Integrating information sources for visualizing Java programs. *IEEE International Conference on Software Maintenance, 2001*, pp.250-258.
27. Lintern, R., Michaud, J., Storey, M., and Wu, X. (2003) Plugging-in visualization: experiences integrating a visualization tool with Eclipse. *The ACM Symposium on Software Visualization, 2003*, p. 47.
28. Voinea, L., Lukkien, J., and Telea, A. (2007) Visual assessment of software evolution. *Science of Computer Programming*, pp. 222-248.
29. Ware, C., Hui, D., and Franck, G. (1993) Visualizing object oriented software in three dimensions. *The IBM Center for Advanced Studies Conference, 1993*, pp. 612-660.
30. Jones, J., Harrold, M., and Stasko, J. (2001) Visualization for Fault Localization. *The ICSE Workshop on Software Visualization, 2001*, pp. 71-75.
31. Balzer, M., Noack, A., Deussen, O., and Lewerentz, C. (2004) Software landscapes: Visualizing the structure

of large software systems. *The ACM Symposium on Software Visualization, 2004*, pp. 261-266.
32. Balzer, M., and Deussen, O. (2004) Hierarchy Based 3D Visualization of Large Software Structures. *IEEE Visualization, 2004.*
33. Balzer, M., Deussen, O., and Lewerentz, C. (2005) Voronoi treemaps for the visualization of software metrics. *The ACM Symposium on Software Visualization, 2005*, pp. 165-172.
34. Löwe, W., and Panas, T. (2005) Rapid Construction of Software Comprehension Tools. *International Journal of Software Engineering & Knowledge Engineering*, 15(6), pp. 905-1023.
35. Wettel, R., and Lanza, M. (2007) Program Comprehension through Software Habitability. *The 15th IEEE International Conference on Program Comprehension, 2007*, pp.231-240.
36. Ham, F. (2003) Using Multilevel Call Matrices in Large Software Projects. *IEEE Symposium on Information Visualization, 2003,* p. 29.
37. Lewerentz, C., and Simon, F. (2002) Metrics-based 3D visualization of large object-oriented programs. *The First International Workshop on Visualizing Software for Understanding and Analysis, 2002*, pp. 70-77.
38. Knodel, J., Muthig, D., Naab, M., and Zeckzer, D. (2006) Towards Empirically Validated Software Architecture Visualization. IESE-Report 071.06/E.