RESEARCH ARTICLE OPEN ACCESS

Software Structural Design Visualization Using Archview Framework

B. Srinivasulu[1], M. Ravi Kumar[2], N. Sirisha[3], P.Srinivas[4]
[1]Department of Computer Science and Engineering, St. Martin's Engineering College, Secundedrabad.
[2]Department of Computer Science and Engineering, Abhinav Hi-tech college of Engineering, RR Dist
[3]Department of Computer Science and Engineering, St.Martin's Engineering College, Secundedrabad.
[4]Department of Information Technology , Nalla Malla Reddy Engineering College, RR Dist.

Abstract
In order to characterize and improve software architecture visualization practice, the paper derives and constructs a qualitative framework, for the assessment of software architecture. The evaluation is used to visualize the relationships between different components. Software Architecture Visualization is used to help all stakeholders to understand the system at all points of the software life cycle. The framework is derived by the application of the Goal Question Metric paradigm to information obtained from literature survey and addresses a number of architectural issues. Solution exists of software architecture visualization is architecture description language, most of the software architecture visualization tools exists are limiting to work in terms of few metrics that are being taken from the software architecture context only.
Keywords: Software architecture, visualization, visualization assessment, methodologies.

I. INTRODUCTION

VISUALIZATION is used to enhance information under-standing by reducing cognitive overload. Using visua-lization tools, people are often able to understand the information presented in a shorter period of time or to a greater depth. The term "visualization" has two connotations. Visualization can refer to the activity that people undertake when building an internal picture about real-world or abstract entities. Visualization can also refer to the process of determining the mappings between abstract or real-world objects and their graphical representation; this process includes decisions on metaphors, environment, and interactivity. This work uses the term "visualization" in the latter sense: the process of mapping entities to graphical representations.

Evaluating a particular visualization technique or tool is problematic. Common practice is that some set of guide-lines is followed and a qualitative summary is produced. As the guidelines may have been used to produce the visualization, there is some bias in such an evaluation. Moreover, these summaries do not usually allow a comparison of competing techniques or tools. A comparison is important because it identifies possible "holes" in the research area or development market. Therefore, for example, a software organization may have the requirement that it needs to visualize their current system with an emphasis on being able to obtain multiple views for multiple users and should also allow querying. Other aspects of the visualization may be less important at this point in time. Thus, a framework for describing the attributes of tools is needed. Once the tools have been assessed in this common framework, a comparison is possible. Such a framework will not be complete and indeed may never be. However, a framework can be used for comparison, discussion, and formative evaluation. In this milieu, we present a frame-work for software architecture visualization evaluation

II. SYSTEM OVERVIEW

The major contribution of this paper is the evaluation framework presented in Section 3. Software architecture visualization evaluation falls into seven key areas: Static Representation, Dynamic Representation, Views, Naviga-tion and Interaction, Task Support, Implementation, and Representation Quality. 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. 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. 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. The framework is used

to evaluate six existing software architecture visualization tools. It is also used to assess tool appropriateness from a variety of stakeholder perspectives. The stakeholder list is extended from that presented in the IEEE 1471 standard. The framework can also be used as design guidelines for an "ideal" tool.

RELATED WORK

This background section briefly surveys the three main areas of the contribution: architecture, visualization, and evaluation.

2.1 Architecture

Architecture can take two roles: one describes how the software systems architecture should be and the other describing how software systems architecture is. Part of the usefulness of architecture analysis is to measure the discrepancy between the prescribed architecture and the architecture that describes the software produced. There are many definitions of architecture. For this work, the IEEE 1471 standard is adopted, where architecture is defined as "the fundamental organization of a system embodied in its components, their relationships to each other and to the environment, and the principles guiding its design and evolution." This is used as the starting definition in this work as it has been agreed upon through a community vetting process. As the frame-work evolved, other aspects, for example, the dynamic aspects of architecture, needed to be incorporated into the framework. For any software system, there are a number of individuals who have some interest in the architecture. These stakeholders have differing requirements of the software architecture depending on the role that they take. The left column in Table 1, from the IEEE 1471 standard , identifies a minimal collection of stakeholders that an architectural description must address communication and understanding of the architecture is essential in ensuring that each stakeholder can play their role during the design, development, and deployment of that software system. Software engineering research has examined the use of specific languages to describe software architecture (see Medvidovic and Taylor's taxonomy. These languages are referred to as Architecture Description Languages (ADLs). Rather than focusing on ADLs for capturing and representing architectural information, the framework presented in this paper is more concerned with the visualization of architectures in the large, whether they have been encoded with an ADL or not. Visualizations may indeed use the paradigm of components and connectors, but this is at a lower level.

2.2 Software Visualization

The most prominent types of visualization defined in the literature are Scientific Visualization, Information Visualization, and Software Visualization. Scientific Visualization is concerned with creating visualizations for physically-based systems, whereas Information Visualization is concerned

III. THE WORKING PRINCIPLE

Before describing the framework itself, the motivation for its development is given. Next, the framework itself is described while indicating the process by which it was derived.

3.1 Motivation for an Architecture Framework

A number of frameworks and taxonomies exist for the evaluation of software visualizations. As software visualization has tended to appeal to its roots in program comprehension, these visualizations are typically concerned with the representation of software at code level, supporting programmers and maintainers. Existing frame-works and taxonomies reflect this focus by looking at low-level areas such as source code, algorithms, and data structures.

The proposed framework will provide a mechanism to discuss key areas and related features of tools and will indicate the trade-offs made by the stakeholders. This is similar to the trade-off technique applied in the cognitive dimensions discussed by Green and Petre in their work on visual programming environments. In supporting developers and maintainers, software visualization has been largely concerned with represent-ing static and dynamic aspects of software at the code level. Architecture visualizations require a larger set of stakeholders. Stakeholders prescribed by IEEE 1471 are general classes of users. For the purpose of software architecture visualization, the list of stakeholders from the left column in Table 1 can be expanded to the list in the right column in Table 1. The extended list on the right in Table 1 illustrates the point that architecture visualization must support a larger number of stakeholders than that supported by traditional software visualization. The right column in Table 1 could also be extended to include other intended stakeholders, such as suppliers, configuration management staff, chief information officers, and auditors.

3.2 Framework Overview

The proposed framework has seven key areas for describing software architecture visualization: Static Representation, Dynamic Representation, Views, Navigation and Interaction, Task Support, Implementation, and Representation Quality. The dimensions identified in the framework are not proposed as a formal representation of the characteristics of software architecture visualizations, but are necessary for discussion about, and evaluation of, such visualizations. Whether they are sufficient is an open question and the subject of future research.

Each of the seven key areas of the proposed framework is discussed in detail below. The

Goal/Metric/Question (GQM) paradigm [1] was used to identify the questions and to then enable the formation of the framework features. GQM was chosen because it defines a measurement model on three levels:

3.3 Relationship to Other Frameworks

The proposed framework has a strong basis in software visualization evaluation. Frameworks and taxonomies such as those by Price et al. Storey et al. and Roman and Cox have been used to categorize and evaluate software visualizations. These works have influenced the creation of the framework. Our approach here is similar to that by Storey et al. "[A framework] can serve several purposes: 1) as a formative evaluation tool... 2) for potential tool users...; and 3) as a comparison tool...". The principal difference is that this work is about architecture, whereas theirs is about development.

Price et al. use a phenomenological approach to derive properties from existing tools, then generalize to a framework. The framework engenders a set of open-ended questions. Our proposed framework attempts to "qualitatively quantify" using an enumeration of possible responses, similar to a Likert scale; such an approach leaves room for judgment on the part of the responder and removes the judgment from the questioner. It is also easier to measure. The measures are qualitative, following Bassil and Keller .

The proposed framework has some degree of overlap with the taxonomy proposed by Price et al. The distinction between Static and Dynamic Representation in this framework has some grounding in the "Data Gathering Time" questions posed by Price et al. Static Representation is concerned with the collection of static elements of the software system (gathered at compile time) and Dynamic Representation is concerned with runtime information. Dynamic Representation also has relationships with Price et al.'s taxonomy in its discussion of "Invasiveness." Ideally, a visualization system should be able to collect data from the target system in such a way that the collection of that data does not change the behavior of that system.

A common theme running throughout both Software Architecture and Software Visualization research is the concept of Multiple Views. Price et al. identify the need for "multiple synchronized views" within visualization, but the proposed framework also considers the view definition, in line with the recommendations of the IEEE 1471 standard.

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. 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?".

Others like 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?

• What questions do they wish to answer through this visualization?

• How can visual metaphors and interaction techniques are used to answer their questions

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.

3.4 Framework Derivation

The primary goal of the proposed framework is to assess system architectures. The framework was derived from an extensive analysis of the literature in the area of software visualization with special emphasis on software architecture. Each of the seven key areas is a conceptual goal which the framework must satisfy. It is this that makes the application of the Goal Question Metric paradigm straightforward.

Rather than describing the complete GQM derivation for each sub goal of the framework, its application in the Static Representation sub goal/key area is demonstrated only. A goal needs a purpose, issue, object, and viewpoint. Thus, here, the need is to assess (the purpose) the adequacy (the issue) of static representation(the object) from the researcher's perspective

(viewpoint). Then, the question "Does the visualization support a multitude of software architectures?" is posed. This process yields the first question in Table 2 and feature SR 1 in Table 3. Continuing in a like manner yields the other three questions in Table 2 and items SR 2-4 in the Static Representation portion in Table 3. Following this process in all key areas provides a straightforward way to generate questions for use in GQM. The metric for the GQM used is the Likert scale with four ordered values plus two nonvalues as this does not overcomplicate the application of the frame-work, and the responses have intrinsic meaning.

These values are summarized in Table 4. The response "Not applicable" (NA) is used where the question is not relevant because the feature is not in the scope of the tool and is different from "No support" (N) in which the scope of the tool would suggest that it should support the feature but it does not. The "Unable to determine" (?) response is used where the question is relevant, but the presence or absence of the feature was not determined.

3.5 Framework Detail

There are some aspects of software

architecture visualization that are not addressed at all in existing software visualization evaluation frameworks. This presents an opportunity to develop a framework for the comparison of such architecture visualizations. The proposed framework is divided into seven key areas. Static representation characterizes the size and accessibility of the architectural information. Dynamic Representation characterizes the support for runtime collection and observation of architectural information. Views characterize the perspective of the observer. Navigation Interaction characterizes the ease of use of the tool. Task Support characterizes the operational use of the visualization. Implementation assesses the suitability of the information for the particular computational environment. Representation Quality characterizes the quality of the information presented to the observer. In the following sections, parenthetical references refer to the leftmost column in Table 3. The intent is to point the discussion of a key area to the embodiment of the feature in the framework by including the GQM questions.

3.5.1 Static Representation (SR)

Static Representation is the architectural information which can be extracted before runtime, for example, source code, test plans, data dictionaries, and other documentation. It is possible that a visualization system will be restricted to a small number of possible architectures. Visualization need not support a multitude of software architectures if that is not the intention of the visualization. In some cases, the software architecture is clearly defined and a single data source exists from which the visualization can take its input. Often, architectural data does not reside in a single location and must be extracted from a multitude of sources. Architecture visualization certainly benefits from the ability to support the recovery of data from a number of disparate sources. Moreover, with multiple data sources, there should be a mechanism for ensuring that the data can be consolidated into a meaningful model for the visualization. Architectural information may not be available directly but is recovered from sources that are non-architectural. For example, file systems may not be directly architecturally related, but they can contain important information that relates to architecture. Even more so, namespaces, modules, classes, methods, and variables can all contribute to a view of the software architecture and, so, a visualization system should support language-specific constructs.

If architectural data is to be retrieved from non architectural data, there is a potential for the data repository to contain large amounts of data from lower levels of abstraction. If this is the strategy employed by the visualization, then the visualization should be able to deal with large volumes of information, that is, the system should be scalable.

3.5.2 Dynamic Representation (DR)

Dynamic Representation is the architectural information that can be extracted during runtime. Some relationships between components of a system will be formed only during execution due the nature of late-binding mechanisms such as inheritance and polymorphism.

Runtime information can indicate a number of aspects of the software architecture. Visualizations should support the collection of runtime information from dynamic data sources in order to relay runtime information. Typically, for smaller software systems, this runtime information will only be available from one source, but, for larger distributed software systems, the visualization may need the capability of recovering data from a number of different sources. These data sources may not reside on the same machine as the visualization system, so the ability to use remote dynamic data sources is useful. Some sources may produce data of one type, where another source produces different data. In this case, the visualization should provide a mechanism by which this data is made coherent. When dynamic events occur, the visualization should be able to display these events appropriately and within the context of the architecture. The visualization must therefore be able to associate incoming events with architectural entities.

3.5.3 Views (V)

Kruchten identifies four specific views of software architecture, whereas the IEEE 1471 standard allows for the definition of an arbitrary number of views. Visualization may support the creation of a number of views of the software architecture and may wish to allow simultaneous access to these views. In the IEEE 1471 standard, architectural views have viewpoints associated with them. A viewpoint defines a number of important aspects about that view, including the stakeholders and concerns that are addressed by that viewpoint, along with the language, modeling techniques, and analytical methods used in constructing the view based on that viewpoint. Visualization may make this information available to the user in order to assist in their understanding of the view they are using.

3.5.4 Navigation and Interaction (NI)

Interactive visualizations systems provide a means by which users will move within, and interact with, the graphical environment. Common user navigation techniques such as panning, zooming, bookmarking, and rotating are usually offered in both 2D and 3D environ-ments. Interaction with the environment can involve selection, deletion, creation, modification, and so on.

An important part of the comprehension process is the formulation of relationships between concepts. Having the ability to follow these relationships is fundamental. Storey et al. indicate that

a software visualization system should provide directional navigation. The visualization should support the user being able to follow concepts in order to gain an understanding of the software architecture.

Searching is the data-space navigation process that allows the user to locate information with respect to a set of criteria. Storey et al. label this as arbitrary navigation—being able to move to a location that is not necessarily reachable by direct links. Sim et al. identify the need for searching architectures for information; so, the visualization should support this searching for arbitrary information.

3.5.5 Task Support (TS)

Task Support is crucial for any usable software visualization system. This area of the framework explores the ability of the visualization to support stakeholders while they are developing and understanding the software architecture. The visualization should support architectural analysis tasks. As comprehension strategies are task dependent, architecture visualizations should support either of top-down or bottom-up strategies, or a combination of the two. An important comprehension task is the identification of anomalies. Architectures may be broken or misused and exhibit unwarranted behavior. The ability to tag graphical elements in visualization is important for various activities. Annotation can allow users to tag entities with information during the formulation of a hypothesis. Visualizations should support any number of stakeholders. In order to facilitate the communication of the architecture to a stakeholder, the visualization must represent the architecture in a suitable manner. Stakeholders may require very different views from other stakeholders. Software architecture can evolve over time. Subsystems may be redesigned; components replaced, new components added, new connectors added, and so on. Architecture visualization should provide a facility to show the evolution. This support may be basic, showing architectural snapshots, or the support may be more advanced by using animation. Visualizations may offer the capability for the users to create, edit, and delete objects in the visualization. In order to be able to fully support the construction of software architecture, the visualization must be able to allow the user to create objects in the domain of the supported viewpoint. Of course, the visualization should also then support the editing and deleting of those objects. Architectural descriptions can be used for the planning, managing, and execution of software development . In order for the visualization to support this task, it should provide rudimentary functionality of a project management tool—or have the ability to communicate with an existing project

3.5.6 Implementation (I)

Visualizations should be able to be generated automatically. If platform choice prohibits remote capture of system data, the visualization should be able to execute on the same platform as the software it is intended to visualize.

3.5.7 Representation Quality (RQ)

Representation Quality is an area of the framework that deals with the capability of the visualization to adequately represent the software architecture. For software architecture visualization, the visualization must present the architecture accurately and represent all of that architecture if the visualization purports to do so. During its execution, software may change its configuration in such a way that its architecture has changed. Software that changes its architecture in such a way is labeled software that has a dynamic architecture. If the visualization is able to support architectural views of the software at runtime, then it may be capable of showing the dynamic aspects of the architecture. In order to do so, the visualization may either support snapshot views of the progression or animate the changes.

3.6 Framework Summary

The two left-hand columns in Table 3 show the outcomes of the application of the GQM paradigm for each key area. The abbreviated key area names in the leftmost column are used in Figs. 1 and 2. The values in the right-hand columns are discussed and developed in Section4.

3.7 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. That is, for the visualization to be deemed useful, it has to provide a means of looking at the different aspects of 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.

3.7.1 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

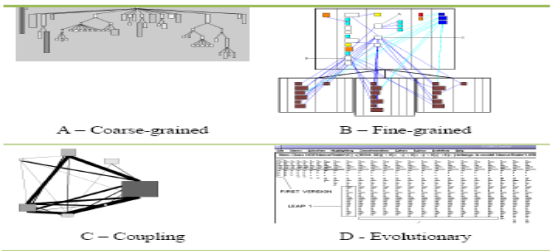

Figure 3 - Four different views for the software architecture by Lanza et al. [23, 24]

3.7.2 Single-view visualization

Single-view visualization was one of the early attempts to visualize multiple aspects of the architecture through a single view. Storey et al. 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. SHriMP did not consider stakeholder communication.

Therefore, 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.

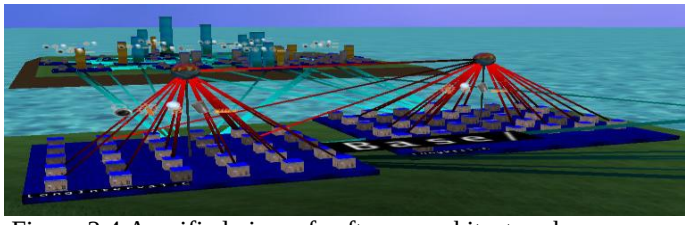

Figure 2.4 A unified view of software architecture by Panas et al.

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.

3.7.3 Virtual Environments 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 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 . The tool provides different interaction modes with zooming and navigation capabilities. Figure 10 shows an instance of visualizing a software architecture using EvoSpaces.

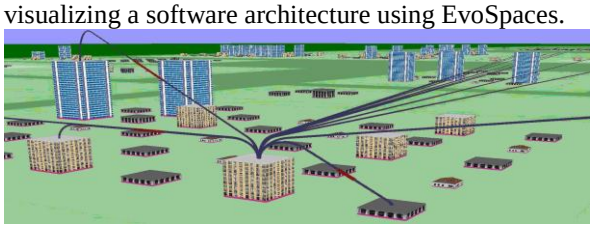

Figure 2.5 - EvoSpaces virtual city by Alam et al.

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.

IV. IMPLEMENTATION OF SYSTEM

4.1 ArchView (AV)

The ArchView tool uses the architecture

analysis activities of extraction, visualization, and calculation. It produces an architecture visualization that presents the use relations in software systems. The relations are stored in a set of files that are read by a browser. The browser reads layout information files and allows the selection of shapes and the manual configuration of layout. A collection of tools is used to manipulate the set of relations to perform selected operations. A VRML generator creates a 3D representation using the 2D layouts and layer position.

4.2 SoftArch (SA)

SoftArch is both a modeling and visualization system for software, allowing information from software systems to be visualized in architectural views. SoftArch supports both static and dynamic visualization of software architecture components and does so at various levels of abstraction. SoftArch's implementation of dynamic visualization is that of annotating and animating static visual forms. SoftArch defines a metamodel of available architecture component types from which software systems can be modeled. In this way, a system's behavior can be visualized using copies of static visualization views at varying levels of abstraction to show both the highly detailed or highly abstracted running.

.

4.4 Ideal Tool

Representing architecture visualization tools through star-plots gives an immediate impression as to the tool's capability. Each tool has its own relative merit and none supports all of the framework's elements and thus represents the trade-offs made by the tool developers. This highlights a potential problem, where an organization may want a single tool to give all stakeholders a central repository for architectural information that can be represented in different ways to each stakeholder. Fig. 2 illustrates a hypothetical tool that combines the features of all tools analyzed under the framework. A salient feature is that this would still not provide full support of all elements of the framework. It is not the direction of this paper to suggest whether or not such a "perfect" tool may be possible to construct. Further, it is undecided whether such a tool is desirable.

V. EXPERIMENTAL RESULTS

The concept of this paper is implemented and different results are shown below.

Fig 5.1 Input of the Software Architecture visualization

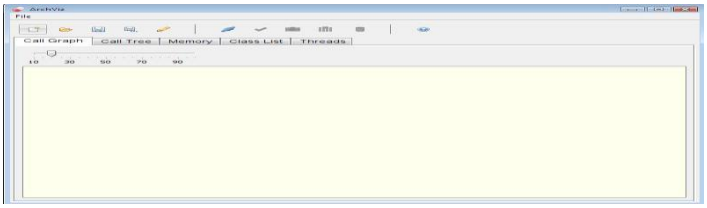

Fig 5.2 Software Architecture Visualization tool home window

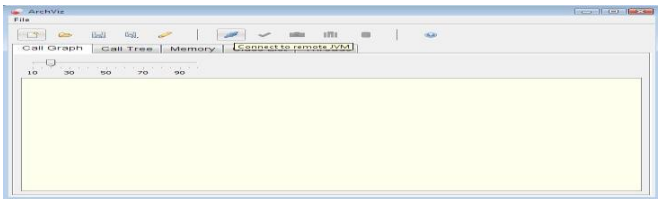

Fig 5.3 ArchViz Connect to JVM

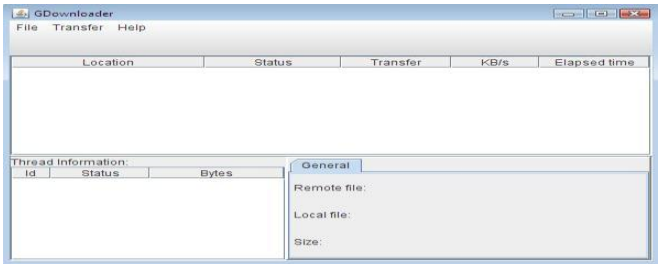

Fig 5.4 Connection between input application and visualization tool

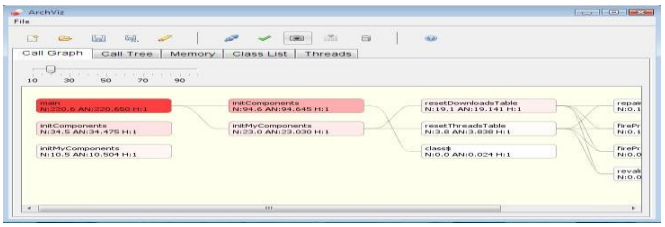

Fig 5.5 Static Call graph of given input application

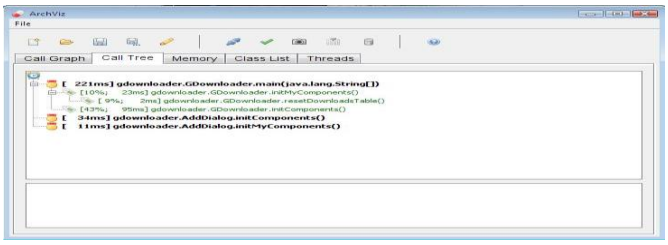

Fig 5.6 Static Call tree of given input application

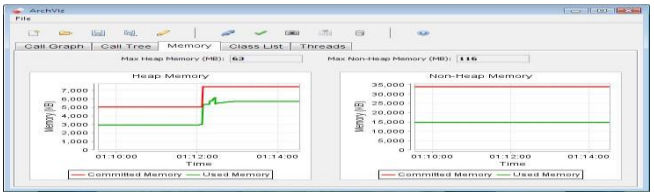

Fig 5.7 Static Memory information of given input application

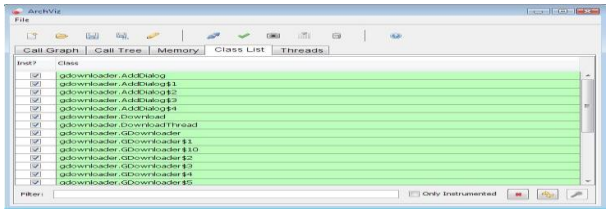

Fig 5.8 Static Class list of given input application

Fig 5.9 Static Threads information of given input application

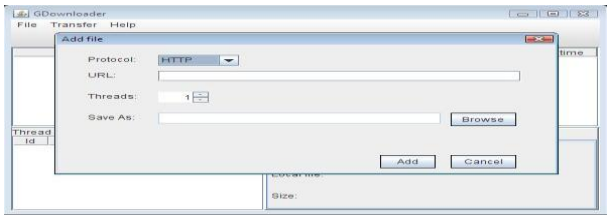

Fig 5.10 Downloading the file using gdownloader

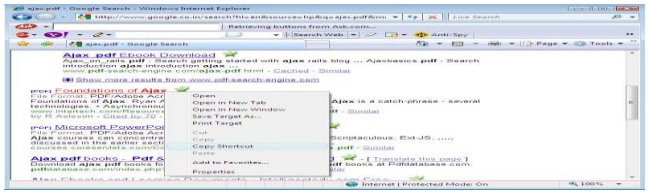

Fig 5.11 Taking the URL link from the net

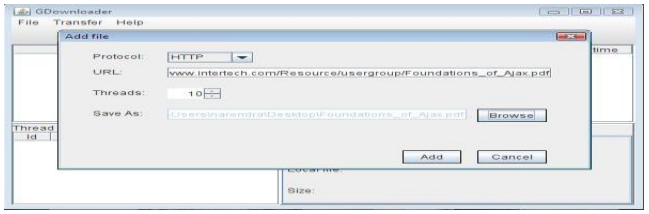

Fig 5.12 Start the downloading file

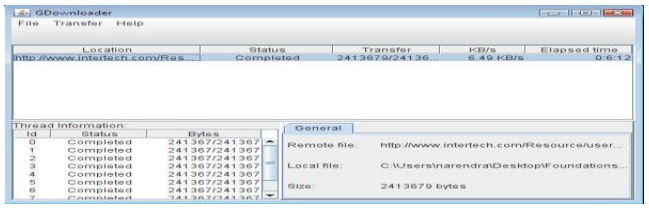

Fig 5.13 Completion of downloading file

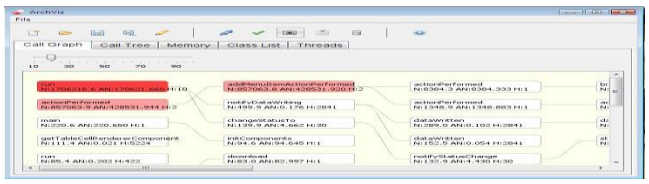

Fig 5.14Dynamic Call graph of given input application

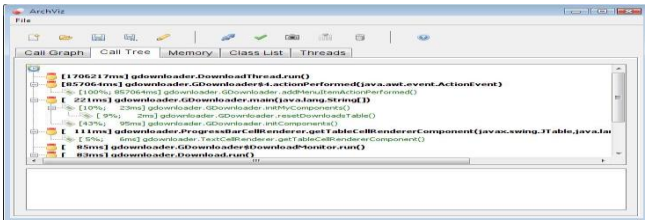

Fig 7.15 Dynamic Call tree of given input application

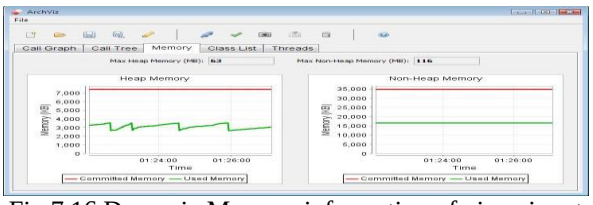

Fig 7.16 Dynamic Memory information of given input application

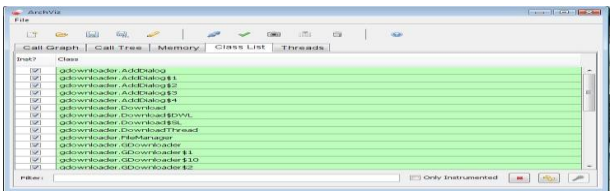

Fig-7.17 Dynamic Class list of given input application

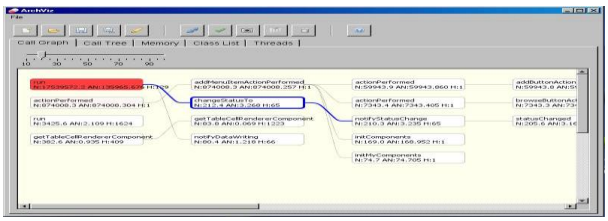

Fig-7.18 Static and dynamic flow between the given input application's methods

VI. CONCLUSION

Software architecture is the gross structure of a system; as such, it presents a different set of problems for visualization than those of visualizing the software at a lower level of abstraction. We have developed and presented a framework for the assessment of the capabilities of software architecture visualization tools and evaluated six tools in this framework. It turns out that no one tool meets all of the criteria of our framework. This is not a bad thing. Moreover, it may be that a one-size-fits-all approach may increase information overload and that a collection of small tools appropriate to each stakeholder's task may be preferable. A side effect of the application of the framework is that it has highlighted features not present in existing tools, for example, Planning and execution and Dynamically changing architecture. These are shown clearly in Fig. 2 and open up the possibility of future research and development.

The issue of the completeness and sufficiency of the framework is an open one and needs to be

addressed by further research. One approach to increase confidence in the framework is by applying it to a larger population of tools. Software engineering theory and practice are evolving, and the notion of software architecture is changing; thus, the definition of software architecture itself will necessarily change. These new developments may give insights into the questions of completeness and sufficiency.

REFERENCES

[1] V. Basili, G. Caldiera, and H.D. Rombach, "The Goal Question Metric Paradigm," Encyclopedia of Software Eng., vol. 2, pp. 528-532, John Wiley & Sons, 1994.

[2] S. Bassil and R. Keller, "A Qualitative and Quantitative Evaluation of Software Visualization Tools," Proc. 23rd IEEE Int'l Conf. Software Eng. Workshop Software Visualization, pp. 33-37, 2001.

[3] S. Card, J. Mackinlay, and B. Shneiderman, Reading in Information Visualization: Using Vision to Think. Morgan Kaufmann, 1999.

[4] A. Eden, "Formal Specification of Object-Oriented Design," Proc. Conf. Multidisciplinary Design in Eng., 2001.

[5] A. Eden, "Visualization of Object-Oriented Architectures," Proc. IEEE 23rd Int'l Conf. Software Eng. Workshop Software Visualization, pp. 5-10, 2001.

[6] A. Eden, "Le PUS: A Visual Formalism for Object-OrientedArchitectures," Proc. Sixth World Conf. Integrated Design and Process Technology, June 2002.

[7] M. Eisenstadt and M. Brayshaw, "A Knowledge EngineeringToolkit: Part I," BYTE: The Small Systems J., pp. 268-282, 1990.

[8] L. Feijs and R. de Yong, "3D Visualization of Software Architectures," Comm. ACM, vol. 41, no. 12, pp. 73-78, Dec. 1998.

[9] K. Gallagher, A. Hatch, and M. Munro, "A Framework for Software Architecture Visualization Assessment," Proc. IEEE Workshop Visualizing Software, pp. 76-82, Sept. 2005.

[10] T. Green, "Instructions and Descriptions: Some Cognitive Aspectsof Programming and Similar Activities," Advanced Visual Interfaces, pp. 21-28, ACM Press, 2000.

[11] T.R.G. Green and M. Petre, "Usability Analysis of Visual Programming Environments: A "Cognitive Dimensions" Framework," J. Visual Languages and Computing, vol. 7, no. 2, pp. 131-174, 1996.

[12] J. Grundy and J. Hosking, "High-Level Static and Dynamic Visualisation of Software Architectures," Proc. IEEE Symp. Visual Languages, pp. 5-12, Sept. 2000.

Author's Profile

Mr. B.Srinivasulu, Post Graduated in Computer Science Engineering (**M.Tech**) From **JNTU**, Hyderabad in **2010 and Graduated in Computer** Science Engineering (B.Tech) from JNTUH, in 2008. He is working as Assistant Professor in Department of Computer Science & Engineering in **St.Martin's Engineering College**, R.R Dist, AP, India. He has 3+ years of Teaching Experience. His Research Interests Include Network Security, Cloud Computing & Data Warehousing and Data Mining.

Mr Mutyala Ravi kumar, Post Graduated in Computer Science & Engineering (M.Tech) , S R M Deemed University,Chennai , 2006, and graduated in Computer Science & Engineering (B.E) From Velagapudi Ramakrishna Siddhartha Engineering college, Vijayawada (Nagarjuna University,Guntur,AP),2001. He is working presently as Senior.Assistant Professor in Department of Computer Science & Engineering in **Abhinav Hi-tech college of Engineering**, RR Dist, A.P, INDIA. He has 7+ years Experience.

Mrs N.Sirisha, Post Graduated in Software Engineering (M.Tech), JNTUH, 2012, and Graduated in Computer Science & Engineering (B.Tech) From JNTU Hyderabad, 2009. She is working presently as an Assitant Professor in Department of Computer Science & Engineering in **St. Martin's Engineering College**, RR Dist, A.P, INDIA. She has 4+ years Experience. Her Research Interests Include Software Engineering, Cloud Computing, Operating Systems and Information Security.

Mr. P.Srinivas, Post Graduated in Computer Science &Engineering (**M.Tech**) From **Lords Institute of Engineering & Technology**, Hyderabad in 2013 and Graduated in Information Technology (B.Tech) form **Nirmala Engineering College**, Manchiriala, 2006. He is working as an Assistant Professor in Department of Information Technology in **Nalla Malla Reddy Engineering College**, Divya Nagar, Near Narapally, Ghatkesar, R.R Dist, AP, and India. He has 6 years of Teaching Experience. His Research Interests Include Software Engineering Data Warehousing and Data Mining, Network Security & Cloud Computing.