Development of the Conceptual Tool for Complete Software Architecture Visualization: DArch (DA)

K. Delhi Babu
Research Scholar
Department of Computer Science
S.V. University, Tirupati,
Andhra Pradesh, India

Dr. P. Govindarajulu
Professor
Department of Computer Science
S.V. University, Tirupati,
Andhra Pradesh, India

A.N. Aruna Kumari
Assistant Professor
Department of Computer Science & Engg.
Sree Vidyanikethan Engineering College,
Tirupati, Andhra Pradesh, India

Summary
Software architecture visualization refers to the process of mapping entities in a software system domain to their graphical representation to aid comprehensive and development. Software visualization can be done based on seven key areas. There are mainly seven visualization tools to satisfy the attributes of seven key areas. Some of the attributes related to dynamic aspects should not supported by the existing tools. In order to support those aspects a new conceptual tool called DArch is proposed. By this we can achieve the properties related to dynamic perspectives. Every tool doesn't support all the attributes related to key areas. So the comprehensive framework was designed to acquire all the key area attributes for complete software architecture visualization.

Key words:
DArch, DA, Architecture Visualizatione.

1. Introduction

Architecture Visualization is the way of understanding the software system clearly and used to enhance information understanding by reducing cognitive overload. Using visualization 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 guidelines 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 environment, we present a framework for software architecture visualization evaluation and over come the limitations of previous tools.

1.1 Result Summary and Contribution

The framework is used to develop the new software architecture visualization tool. It helps to visualize software architecture completely. It is also used to assess tool appropriateness from a variety of stakeholder perspectives. The framework can also be used to design an "ideal" tool, for visualization of software architecture through all key attributes.

2. 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 describing how the software system's architecture should be and the other describing how a software system's architecture is. In section 3.1.describing how a software system's

Manuscript received April 5, 2009
Manuscript revised April 20, 2009

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 [6], [9], [22].For this work, the IEEE 1471 standard [15] 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 framework 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 [15], 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 [19]). These languages are referred to as Architecture Description Languages (ADLs). Rather than focusing on ADLs for capturing and representing architectural information, the framework presented is more concerned with the visualization of architectures in the large, whether they have been encoded with an ADL or not [10]. Visualizations may indeed use the paradigm of components and connectors, but this is at a lower level.

TABLE 1 Stakeholders

IEEE-1471 Stakeholders	*Extended Stakeholders*
Users	Users
Acquirers	Acquirers
Developers	Developers
Maintainers	Maintainers
	Architects
	Operators
	Testers
	Designers
	Development managers
	Sales and field support
	System administrators

2.2 Software Visualization

The most prominent types of visualization defined in the literature are

1. Scientific Visualization
2. Information Visualization
3. Software Visualization

Scientific Visualization is concerned with creating visualizations for physically-based systems. Information Visualization is concerned with abstract nonphysical data.

Software Visualization has been defined as a discipline that makes use of various forms of imagery to provide insight and understanding and to reduce complexity of the existing software system under consideration.

The motivation for visualizing software is to reduce the cost of software development and its evolution. Software visualization can support software system evolution by helping stakeholders to understand the software at various levels of abstraction and at different points of the software life cycle. Software Visualization can be seen as the application of Information Visualization techniques to software, as the data collected from all areas of a system development, such as code, documentation, and user studies, is abstract and, hence, has no associated physical structure.

Software Visualization is the process of mapping entities in a software system domain to graphical representations to aid comprehension and development. It has traditionally been focused on aiding the understanding of software systems by those who perform development and maintenance tasks on that software. Although Software Visualization supports the software development and maintenance process, this focus excludes other valid stakeholders such as Users and Acquirers as listed in Table 1. Software Architecture Visualization can help all stakeholders to understand the system at all points of the software life cycle.

2.3 Evaluating Software Visualizations

A number of taxonomies have been developed for classifying software visualizations. Taxonomies define a number of features that visualizations can be measured against. A commonly used method for evaluating software visualizations is to apply these taxonomies as an evaluation framework. Price et al. [20] present a taxonomy of Software Visualization with six distinct categories: Scope (the range of systems that can be visualized, platform for system, and scalability), Content (the subset of data from Scope that is actually used in the visualization: control flow, data flow, and algorithms), Form (the characteristics of the visualization: medium, level of detail, and synchronized views),Method (how the data for the visualizations is gathered: automatically generated visualization, code instrumentation, and noninvasive probes), Interaction (user interaction and control: use of buttons and menus and navigation), and Effectiveness (how well the visualizations meet their objectives: purpose of the visualizations, clarity, and degree of empirical evaluation). These categories are structured hierarchically, with each category expanded into subcategories. The categories were derived bottom-up, first by surveying existing taxonomies, then examining current tools, and finally letting these observations suggest a new formulation.

Bassil and Keller [17] use Price et al.'s framework to qualitatively analyze a collection of softwarevisualization tools. Maletic et al. [18] enhance the Price framework with regard to task orientation. Task orientation is similar to our use of stakeholders; however, we have a larger scope of task than that presented by Maletic et al.

3. Evaluation Framework

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 [20], [1], [7]. 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 frameworks and taxonomies reflect this focus by looking at low-level areas such as source code, algorithms, and data structures [5], [12], [20], [26]. 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 [12] in their work on visual programming environments.

In supporting developers and maintainers, software visualization has been largely concerned with representing 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 Table1 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 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 [21] straightforward.

Rather than describing the complete GQM derivation for each subgoal of the framework, its application in the Static Representation subgoal/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 framework, and the responses have intrinsic meaning.

3.3 Framework Details

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.
We can evaluate the visualization tools based on the key areas that are required in the derivation of the framework. Evaluating the visualization tools based on key areas

TABLE2

		AV	SB	SA	SoFi	LePUS	EA	Avis	DA
	Key Area : Static Representation (SR)								
SR1	Multiple software rchitectures	Y	Y	Y	Y	Y	Y	Y	N
SR2	Types of software architecture	Y	Y	Y	Y	Y	Y	Y	N
SR3	Recovery of software Architecture information	Y?	Y?	N	Y?	NA	N	Y	N
SR4	Accommodate large volumes of information	Y	?	N?	Y	NA	Y	Y	N
	Key Area : Dynamic Representation (DR)								
DR1	Support dynamic data	N	N	N?	N	NA	N	Y?	N
DR2	Associate events with architectural elements	N	N	N?	N	NA	N	Y	N
DR3	Non invasive approaches	N	N	N?	N	NA	N	Y	Y
DR4	Live collection	N	N	N?	N	NA	N	Y	N
DR5	Replay data	N	N	N?	N	NA	N	Y	N
	Key Area : Views (V)								
V1	Multiple Views	N	Y	Y	N	Y	Y	Y	N
V2	Representation of viewpoint Definition	N	Y?	N?	N	N	Y	Y?	N
	Key Area : Interaction and Navigation (NI)								
NI1	Browsing	Y	Y	Y	N?	NA	Y	Y	N
NI2	Searching	N	Y	N	N	NA	Y	Y	N
NI3	Query drilling	N	N	N	N	NA	Y	Y	N
NI4	Inter-View navigation	N	Y	Y	N	NA	Y	Y	N
NI5	View navigation	Y	N?	Y	N	NA	Y	Y	N
	Key Area : Task Support (SR)								
TS1	Represent anomalies	Y	?	N	Y	NA	N	N?	N
TS2	Comprehension	Y	Y	Y	Y	Y	Y	Y	N
TS3	Annotation	N	N	Y	N	Y?	Y	Y?	N
TS4	Communication	Y	Y	Y	Y	Y	Y	Y	N
TS5	Show evolution	N	N?	N	N	Y?	N?	N	Y
TS6	Construction	N	N	Y	N	Y	Y	N?	N
TS7	Planning and Execution	N	N	N	N	N	N	Y?	Y
TS8	Evaluation	Y	Y?	N	Y	Y	Y	Y?	N
TS9	Comparison	Y?	Y?	N	Y?	Y	N?	N	N
TS10	Show rationale	N	N?	N	N	N?	Y	Y?	Y
	Key Area : Implementation (I)								
I1	Automatic generation	Y	Y	N	Y	NA	N	Y	N
I2	Platform dependence	Y?	N?	Y	?	NA	N	Y	N
I3	Multiple users	Y	Y	Y	Y	Y	NA	Y?	N
	Key Area : Representation Quality (RQ)								
RQ1	High fidelity and completeness	Y	Y	Y	Y	NA	Y	Y?	N
RQ2	Dynamically changing nature	N	N	N	N	NA	N	N?	Y

TABLE 3: Possible Responses to the items in table2

Response	Meaning
Y	Full support
Y?	Mainly supported
N?	Mainly not supported
N	No support
NA	Not Applicable (not in scope)
?	Unable to determine

3.3.1 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. A Visualization need not support a multitude of software architectures if that is not the intention of the visualization. (SR 1: Does the visualization support a multitude of software architectures?) 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. (SR 2: Does the visualization support the appropriate types of static software architecture data sources?) An 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 nonarchitectural. (SR 3: Does the visualization support the recovery of architectural information from sources that are not directly 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 nonarchitectural data, there is a potential for the data repository to contain large amounts of data from lower levels of abstraction. (SR 4:Can the visualization accommodate large amounts of architectural data?) 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.

3.3.2 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. (DR 1:Does the visualization support an appropriate set of dynamic data sources?) 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. (DR 2:Does the visualization support association of dynamic events with elements of the software architecture, during execution of the software?) The visualization must therefore be able to associate incoming events with architectural entities.
Any method of recording dynamic information from a software system will affect that software system in some way. (DR 4:Does the visualization allow live collection of dynamic data?) At one extreme, there is the directly invasive approach of adding lines to the software source code. At the other extreme, there is retrieval of information from a virtual machine. The visualization system should support a suitable approach to recovery of dynamic architecture data in the least invasive way; disruptive behavior is not desirable. (DR 3:Does the visualization support noninvasive collection of dynamic data?)
By visualizing the dynamic data as it is generated, there may be an affect on the software being visualized. A "postmortem style" has the benefit of knowing the period of time over which the visualization occurs. This is useful to a visualization in that it can render a display for a particular instance in time while knowing what will occur next. (DR 5: Does the visualization allow recording of dynamic data for subsequent replay?)

3.3.3 Views (V)

Kruchten [17] identifies four specific views of software architecture, whereas the IEEE 1471 standard allows for the definition of an arbitrary number of views. (V 1:Does the visualization allow for multiple views of software architecture?)A 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. (V 2:Does the visualization display a representation of the viewpoint definition?) A visualization may make this information available to the user in order to assist in their understanding of the view they are using.

3.3.4 Navigation and Interaction (NI)

Interactive visualizations systems provide a means by which users will move within, and interact with, the graphical environment. (NI 1: Can users browse the visualization by following concepts?) Common user navigation techniques such as panning, zooming, book marking, and rotating are usually offered in both 2D and 3D environments. 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. [7] 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. (NI 2: Can users search for arbitrary architectural information?) Storey et al. [7] label this as arbitrary navigation—being able to move to a location that is not necessarily reachable by direct links. Sim et al. [24] identify the need for searching architectures for information; so, the visualization should support this searching for arbitrary information.
Query drilling is a term that describes a method of dataspace navigation that is a particular hybrid of browsing and searching. (NI 3:Can the user query-drill architectural information?) It allows a user to search the data space and then recursively search within the resulting data set.
Architecture is often comprised of a number of views. Moving between views is essential in order to understand an architecture from different viewpoints. (NI 4: Can users navigate between views?) Context should also be maintained when switching between views so as to reduce disorientation. Along with data-space navigation, the movement within a view is also important. Shneiderman's mantra for visualization is overview first, zoom, and filter, and then show details on demand [23]. A visualization system should support this strategy. Also, the visualization should allow the user to move around so as to focus on and see the information they are looking for. Typical navigational support would be pan and zoom. While allowing the user to navigate, the visualization should provide orientation clues in order to reduce disorientation. (NI 5:Can users navigate appropriately within a view?)

3.3.5 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.(TS 2:Does the visualization support software architectural comprehension?) An important comprehension task is the identification of anomalies. Architectures may be broken or misused and exhibit unwarranted behavior. (TS 1: Does the visualization support the representation of anomalies?) The ability to tag graphical elements in a visualization is important for various activities. Annotation can allow users to tag entities with information during the formulation of a hypothesis. (TS 3: Does the visualization support annotation?) 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. (TS 4: Does the visualization support the communication of the architecture to intended stakeholders?) 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. (TS 5:Does the visualization show the evolution of software architecture?) An 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.(TS 6:Does the visualization support construction of software architectures?) 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 [15]. 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 management tool. (TS 7: Does the visualization support software planning and development?)

Software architecture evaluation allows the architects and designers to determine the quality of the software architecture and to predict the quality of the software that conforms to the architecture description [15]. To support this, a visualization should have some mechanism by which quality descriptions can be associated with components of the software being visualized. (TS 8: Does the visualization support evaluation of software architectures?) A typical use of software architecture visualization is the comparison of as-implemented with as-designed architecture. The visualization should be able to support the display of these two architectures and allow users to make meaningful comparisons between them. (TS 9: Does the visualization support the comparison of software architectures?) Software built from a software product line is a typical scenario where comparison of architectures is particularly useful.

The rationale for the selection of architecture and the selection of the individual architectures of the components of that architecture are included in architectural descriptions. (TS 10: Does the visualization represent rationale?)Rationale can also be associated with each viewpoint of an architecture. By showing the rationale for the elements of the architecture and the architecture as a whole, a visualization will allow a user to have an insight into the decision making process.

3.3.6 Implementation (I)

Visualizations should be able to be generated automatically. (I 1:Can the visualization 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. (I 2: Can the visualization be executed on the platform of the target system?) Where possible, remote capture may be preferred for its potential in reducing unwanted interaction with the software. As there are many stakeholder roles in a software system, there may also be a one-to-one mapping of role to physical users. Therefore, the visualization should support multiple users concurrently or asynchronously. (I 3: Does the visualization support multiple users?)

3.3.7 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. (RQ 1: Does the visualization achieve high fidelity and completeness?) 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. (RQ 2 : Does the visualization support the representation of dynamically changing software architecture?) In order to do so, the visualization may either support snapshot views of the progression or animate the changes.

4. Software Architecture visualization tools

Each visualization tool can satisfy some specific activities. Only one tool does not satisfy the needs to visualize the software completely and effectively.

4.1.1 ArchView (AV)

The ArchView [9] 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.

4.1.2 The Searchable Bookshelf (SB)

The Searchable Bookshelf [24] visualization attempts to combine both searching and browsing approaches to software comprehension. The Searchable Bookshelf adds search capabilities to the Software Bookshelf. Users can browse the software structure from an initial overview by navigating through an HTML style display and a software landscape central view. Here is an example of the difference between searching and query drilling. The Searchable Bookshelf allows searching but does not allow extended searching within the resulting data space.
This visualization affords the user a number of different views; however, the number of views is limited and the user cannot add custom views. Dynamic data is not linked to the static representations of the architecture. The visualization is therefore unable to deal with architectures that change configuration during runtime.

4.1.3 SoftArch (SA)

SoftArch [13] 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 system information. SoftArch is integrated into a development environment; thus, it addresses a key criticism of other visualizations: It provides a mechanism by which it can be used by developers during software development. Other aspects of architecture such as project management, architecture comparison, and architecture evaluation are not directly supported in SoftArch.

4.1.4 SoFi

SoFi [4] is a tool that performs source code analysis in order to compare intended architecture with implemented architecture. SoFi's clusters source files into a structure based on source file naming schemes. SoFi relies heavily on intervention by an architect to perform restructuring. This restricts the applicability of this visualization to scenarios that require automated generation of a visualization of an existing system. SoFi is focused on lower level areas of architecture and does not support dynamic data. Visualizing evolution can only be supported by repeated application of the tool and visually comparing the differences between subsequent images.

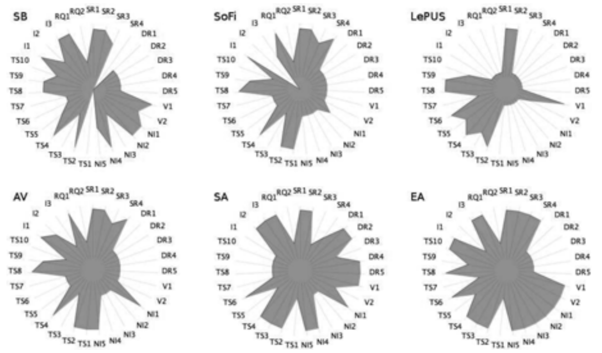

Fig.1: Starplots of visualization tools

4.1.5 LePUS

LePUS is a formal language dedicated to the specification of object-oriented design and architecture [5], [6], [7]. LePUS diagrams are intended to be used in the specification of architectures and design patterns and in the documentation of frameworks and programs. As a visual language, LePUS is not concerned with the extraction of architectural information from systems but is simply a means by which an architect can encode software architecture for communication to other stakeholders in that architecture. This will allow for some activities, such as construction, evaluation, and comparison, but is not suited to core visualization activities such as searching and query drilling.

4.1.6 Enterprise Architect (EA)

Enterprise Architect [25] is a UML CASE tool that allows software architects, designers, and analysts to design software from several viewpoints. EA can be used from requirements capture to UML modeling to testing and project management. EA utilizes a graphical user interface that sits above an entity-relationship repository. The primary mechanism for modeling software systems in EA is to use diagrams. Entity templates are dragged onto a diagram area, causing a new entity to be created. These entities can be edited using the graphical user interface. Links can be formed between diagram entities. These links cause relationships to be formed between entities in the underlying model. Existing entities can be dragged onto newly formed diagrams and any existing relationships are automatically shown. Thus, the entity-relationship model is distinct from the visual representations that form the user- interface. EA's primary use is for designing new software but it also offers a broad range of other tools. For example, EA also allows existing software to be parsed and imported. EA supports many activities and is suited to a wider audience of stakeholders. It does not support dynamic data and has difficulty in showing architectural evolution. EA does permit the construction of new views.

4.1.7 Arch Vis (Avis)

Arch Vis is prototype software architecture visualization tool. Its design was driven by the key concerns regarding software architecture visualization requirements. That is to say that Arch Vis was designed and built using the evaluation framework as requirements. In this sense, including it in this list is skews the results. However, the framework and Arch Vis were developed in parallel, so features were added to the framework after the design of Arch Vis was complete. Figure 2 shows star plot of Arch Vis visualization tool.

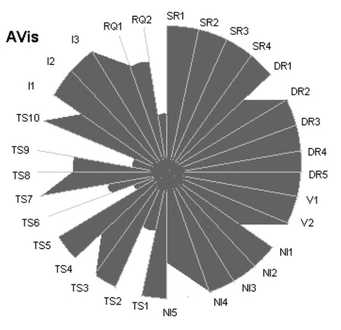

Figure 2

All these seven existing visualization tools all the attributes that are present in the seven key areas. We can know this by superimposing the starplots of all the existing tools on one another we can obtain the combined starplot of all the existing tools. This combination starplot clearly shows that some of the attributes related to dynamic events should not supported by the existing tools. The figure 3 shows the combined starplot of all the visualization tools. In this representation we can find that some specific activities should not satisfied by the existing tools. In order satisfy those activities we can propose a new tool for visualizing the software completely. In order to satisfy all the attributes related to dynamic events we can propose a new tool, it can be referred as DArch(DA).

4.1.8. DArch (DA)

The proposed conceptual tool by us covers the activities of non invasive collection of data, evolution of software, planning and development, rationale selection of architectures and dynamically changing architecture. This tool is mainly focused on the dynamic events that are related to a particular software development. By utilizing the new tool we can retrieve the data required for visualizing the software architecture in a proper way in order to avoid abnormal behavior. Figure 4 shows the star plot of the proposed conceptual tool DA.

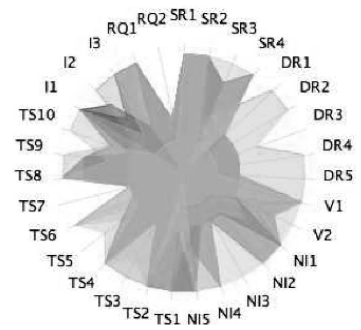

Figure 3

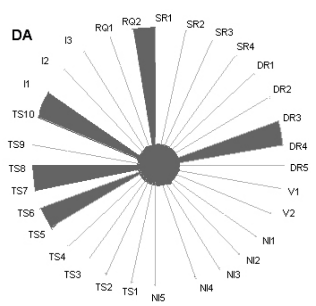

Figure 4

5. Evaluations

Table2 presents the evaluation of the features in the tabular form. Most tools do reasonably well in static representation Dynamic representation is another matter, as none of the surveyed tools have support for this key area. Most tools support multiple views; none support viewpoint definition .Navigation and interaction is supported by browsing in all tools. The Enterprise Architect is the only tool that has all the searching, querying and view navigation features. It shows that all the tools are deficiet in task support. This is mildly surprising as one would expect architecture tools to be closely allied with project management and IDE systems. It is also surprising to note that not all tools have automatic generation and multiple user support. And all tools support high fidelity visualizations, but none dynamically changing architectures.

With respect to Arch Vis, it is worth nothing that it does not meet the full set of requirements. It does not show evolution and give comparisons, and has only lightweight support for anomalies and construction. It does meet the dynamic representation criteria, and thus has one singular advantage over all the other tools.

6. Ideal Tool

Representing architecture visualization tools through starplots 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.

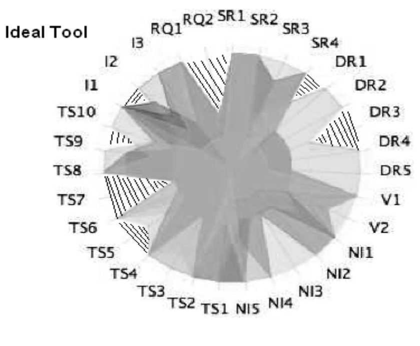

Figure 5

The below figure illustrates an ideal tool that combines the features of all tools analyzed under the framework. A salient feature is that this would provide full support of all elements of the framework. It is the direction of this paper to suggest such a "perfect" tool may be possible to construct. In the figure the lined portion indicates the support for the new tool called DArch (DA). By including this tool along with the existing tools we can meet the all requirements to achieve an ideal tool in order to satisfy all the attributes related to the seven key areas discussed above.

7. 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 seven 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 (TS 7) and Dynamically changing architecture (RQ 2). These are shown clearly in Fig. 3. Thus, we are using the framework to define and prototype an architecture visualization tool [14]. It seems clear that such a tool will need to be tailorable to the specific stakeholder in order to be of any practicable use. By inducing the new comprehensive tool along with the existing tools we can visualize the software completely.

References

[1] G-C. Roman and K.C. Cox, "A Taxonomy of Program Visualization Systems," Computer, vol. 26, no. 12, pp. 11-24, Dec. 1993.

[2] P. Kruchten, "The 4 + 1 View Model of Software Architecture," IEEE Software, vol. 12, no. 6, pp. 42-50, Nov. 1995.

[3] M. Storey, D. Cubranic, and D. German, "On the Use of Visualization to Support Awareness of Human Activities in Software Development: A Survey and Framework," Proc. ACM Symp. Software Visualization, pp. 193-202, 2005.

[4] I. Carmichael, V. Tzerpos, and R. Holt, "Design Maintenance: Unexpected architectural Interactions," Proc. Int'l Conf. Software Maintenance, pp. 134-137, 1995.

[5] T. Green, "Instructions and Descriptions: Some Cognitive Aspects of Programming and Similar Activities," Advanced Visual Inter-faces, pp. 21-28, ACM Press, 2000.

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

[7] M. Storey, F. Fracchia, and H. Muller, "Cognitive Design

Elements to Support the Construction of a Mental Model During Software Exploration," J. Systems and Software, vol. 44, pp. 171-185, 1999.
[8] M. Eisenstadt and M. Brayshaw, "A Knowledge Engineering Toolkit: Part I," BYTE: The Small Systems J., pp. 268-282, 1990.[9] L. Feijs and R. de Yong, "3D Visualization of Software Architectures," Comm. ACM, vol. 41, no. 12, pp. 73-78, Dec. 1998.
[10] K. Gallagher, A. Hatch, and M. Munro, "A Framework for Software Architecture Visualization Assessment," Proc. IEEE Workshop Visualizing Software, pp. 76-82, Sept. 2005.
[11] A. Eden, "Formal Specification of Object-Oriented Design," Proc.Conf. Multidisciplinary Design in Eng., 2001.
[12] 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.
[13] J. Grundy and J. Hosking, "High-Level Static and Dynamic Visualisation of Software Architectures," Proc. IEEE Symp. Visual Languages, pp. 5-12, Sept. 2000.
[14] A. Hatch, "Software Architecture Visualisation," PhD dissertation, Univ. of Durham, 2004.
[15] "IEEE Recommended Practice for Architectural Description of Software Intensive Systems," technical report, IEEE, 2000.
[16] C. Knight and M. Munro, "Visualising Software—A Key Research Area," Proc. Int'l Conf. Software Maintenance, p. 436, 1999.
[17] 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.
[18] J. Maletic, A. Marcus, and M. Collard, "A Task Oriented View of Software Visualization," Proc. IEEE Workshop Visualizing Software for Understanding and Analysis, pp. 32-40, 2002.
[19] N. Medvidovic and R. Taylor, "A Classification and Comparison Framework for Software Architecture Description Languages," IEEE Trans. Software Eng., vol. 26, no. 1, pp. 70-93, Jan. 2000.
[20] B.A. Price, R. Baecker, and I.S. Small, "A Principled Taxonomy of Software Visualization," J. Visual Languages and Computing, vol. 4, no. 3, pp. 211-266, 1993.
[21] 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.
[22] M. Shaw and D. Garlan, Software Architecture: Perspectives on an Emerging Discipline. Prentice Hall, 1996.
[23] B. Shneiderman, Designing the User Interface: Strategies for Effective Human-Computer Interaction. Addison-Wesley, 1998.
[24] S. Sim, C. Clarke, R. Holt, and A. Cox, "Browsing and Searching Software Architectures," Proc. Int'l Conf. Software Maintenance, pp. 381-390, Sept. 1999.
[25] Sparx Systems, Enterprise Architect, http://www.sparxsystems. com.au, 2008.
[26] J. Stasko and C. Patterson, "Understanding and Characterizing Program Visualization Systems," Proc. IEEE Workshop Visual Languages, pp. 3-10, 1992.
[27] S. Card, J. Mackinlay, and B. Shneiderman, Reading in Information Visualization: Using Vision to Think. Morgan Kaufmann, 1999.
[28] A. Eden, "Le PUS: A Visual Formalism for Object-Oriented Architectures," Proc. Sixth World Conf. Integrated Design and Process Technology, June 2002.