A Heuristic Approach for Selection of Software Architecture Using Functional Key Areas and Visualization Tools Integrated with DArch

K. Delhi Babu
S.V. University,
Tirupati

Dr. P. Govindarajulu
S.V. University,
Tirupati

Dr. A. Ramamohana Reddy
S.V. University,
Tirupati

A.N. Aruna Kumari
Sree Vidyanikethan
Engineering College, Tirupati

Abstract
Architecting the distributed software applications is a complex design activity. The selection of a best design among number of design alternatives is an important activity. To satisfy various stakeholders' functional and non-functional requirements of a particular application, there is a need to take a number of decisions. This problem has become the multiple decision making problem. Visualization tools based on functional key areas integrated with DArch have been used in the context. In this paper we are proposing a new approach for selection of Software Architectures using functional key areas which are basis for visualization tools.

Key words:
Heuristic, DArch.

1. Introduction

In general the software development organizations face the problem of selecting the best design from a group of designs alternatives. Architecting the systems like distributed software is a complex design activity. It involves making decisions about a number of inter-dependent design choices that relate to a range of design concerns. Each decision requires selecting among a number of alternatives; each of which impacts differently on various quality attributes. Additionally, there are usually a number of stakeholders participating in the decision-making process with different, often conflicting, quality goals, and project constraints, such as support dynamic data, Platform dependence, accommodate large volumes of information etc. This technique can help all stack holders to understand the system in functional key areas perspective.

2. Related work

2.1 Software Architecture Evaluation Techniques

Software quality is the degree to which an application possesses the desired combination of quality attributes**.** Software architecture evaluation has emerged as an important software quality assurance technique. The principle objective of evaluating architecture is to assess the potential of the chosen architecture to deliver a system capable of fulfilling required quality requirements. A number of methods, such as Architecture Tradeoff Analysis Method (ATAM) [3] and Architecture-Level Maintainability Analysis (ALMA) [4], have been developed to evaluate the quality related issues at the architecture level. The architecture design evaluation methods like Quality Attribute Workshop [5], Cost-Benefit Analysis Method [2], Active Reviews for Intermediate Designs [6] and Attribute-Driven Design [7] includes a number of activities that logically belong to different parts of the traditional SDLC [8]. Kazman et al. [2] propose the Cost Benefit Analysis Method **(CBA**M) to quantify design decision in terms of cost benefit analysis. ATAM is a SA evaluation method, which itself needs a **SA** as an input to the evaluation process.

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.

Manuscript received June 5, 2009
Manuscript revised June 20, 2009

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

2.3 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. [17] 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 [24] use Price et al.'s framework to qualitatively analyze a collection of software visualization tools. Maletic et al. [16] 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. The seven functional key areas for any software architecture are described [9] below.

2.3.1 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. 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. 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. 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. 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.

2.3.2 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 to 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. Any method of recording dynamic information from a software system will affect that software system in some way. 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. 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.

2.3.3 Views (V)

Kruchten [10] identifies four specific views of software architecture, whereas the IEEE 1471 standard allows for the definition of an arbitrary number of views. 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. A visualization may make this information available to the user in order to assist in their understanding of the view they are using.

2.3.4 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, 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. [12] 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. [12] label this as arbitrary navigation—being able to move to a location that is not necessarily reachable by direct links. Sim et al. [19] 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. 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. 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 [18]. 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.

2.3.5 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. 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 a 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. 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. 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. 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. 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. Software built from a softwareproduct 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. 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.

2.3.6 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. 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.

2.3.7 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. 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.

3.0 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.

3.1.1 Arch View (AV)

The ArchView [23] 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.

3.1.2 The Searchable Bookshelf (SB)

The Searchable Bookshelf [19] 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.

3.1.3 SoftArch (SA)

SoftArch [14] 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.

3.1.4 SoFi

SoFi 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.

3.1.5 LePUS

LePUS is a formal language dedicated to the specification of object-oriented design and architecture [11], [12], [13]. 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.

3.1.6 Enterprise Architect (EA)

Enterprise Architect [20] 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.

3.1.7 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.

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. In this representation we can find that some specific activities should not be satisfied by the existing tools. In order to 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).

3.2 DArch (DA)

The proposed conceptual tool [22] 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 3.2 shows the star plot of the proposed conceptual tool DA.

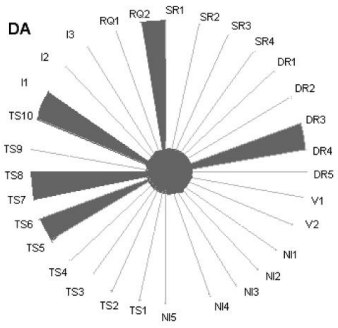

Fig. 3.2

3.3 Selection of Software Architecture based on functional key areas using visualization tools

Visualization Technique as a critical decision making tool for several applications. There are several visualization tools available for a particular software architecture visualization. The principle difference is that this work is about selecting software architecture among alternatives based on functional key area attributes. Where as the visualization tools are to visualize a particular architecture only.

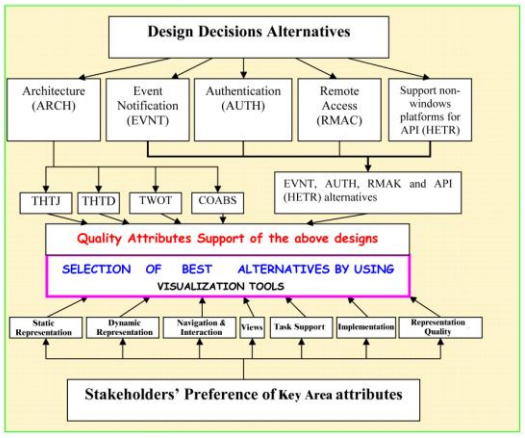

Figure 3.3: A Framework for selection of software architecture based on functional key area attributes using visualization tools

This tents to neglect the solution development stage in a decision making process, the implications of intermediate decisions and analysis are lost. Tradeoffs with a design alternative tend to be much less explicit. This holistic view may lead to situations where the preferences are given for functional key area attributes hinges on sensitive and critical decisions of which stakeholders are not aware. Fig 3.3 shows this model of selecting software architectures.

4.0 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.

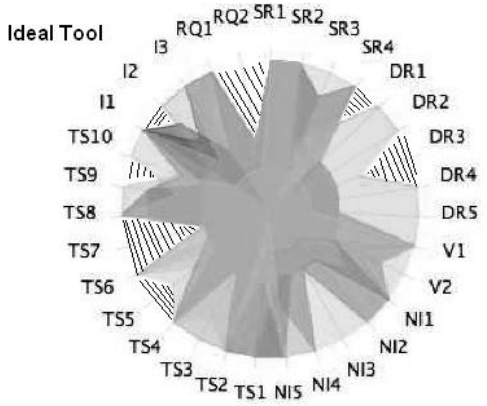

Fig. 4.0

The below figure 4.0 illustrates an ideal tool that combines the features of all tools analyzed. A salient feature is that this would provide full support of all attributes of the key functional areas. 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 with help of this ideal tool we can select best architecture among various alternatives by visualization.

4.1 Selection of Software Architectures based on functional key areas using visualization tools integrated with DArch

The Fig.4.1 is the framework [21] using ideal tool of architecture visualization for selection of required software architecture using functional key area attributes [22].

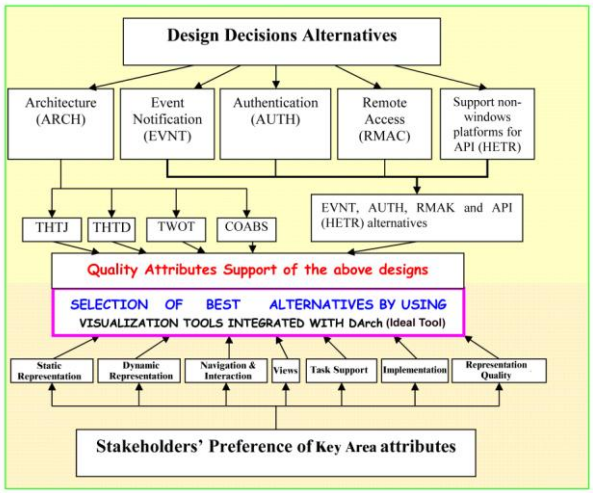

Figure 4.1 A Framework for selection of software architecture based on visualization tools integrated with DArch (Ideal Tool)

Various design decisions are used to design various architectures of various problems. This model used to select the best architecture suitable for stakeholders functional requirements just by visualization.

5.0 Conclusion

In the selection of software architecture among various alternatives technological justification involves active participation of different groups of specialists (stakeholders). It is absolutely necessary to have their preferences. So this model is helpful to visualize their required functional priorities in various software architecture alternatives. It helps to reduce the cost, effort, time etc. of software development. This model also used to select suitable software architecture best suitable for a specific problem and also for some applications with limited modifications. We have developed and presented a model for the complete visualization of the functional capabilities of software architecture using an ideal tool [22]. These issues lead to an architecture better prepared for future change.

Reference

[1] Chung L et al, "Non-Functional Requirements in Software Engineering": Kluwer Academic Publishers, Boston, MA. 1999].

[2] Kazman, Rick; Asundi, Jai; & Klein, Mark. "Quantifying the Costs and Benefits of Architectural Decisions," 297-306. Proceedings of the 23rd International Conference on Software Engineering (ICSE 2001). Toronto, Ontario, Canada, May 12 - 19, 2001. Los Alamitos, CA: IEEE Computer Society, 2001.

[3] Kazman, R., Barbacci, M., Klein, M., and Carriere, J 1999. "Experience with performing architecture tradeoff analysis". In Proceedings of the 21st International Conferences on software Engineering (ICSE'99), pp.54-63.

[4] Bengtsson, P., Lassing. N., Bosch, J., and Vliet, H.V. 2004. "Architecture-level modifiability analysis (ALMA)". Journal of Systems and Software 69(1/2): 129-147.

[5] Barbacci, Mario R.; Ellison, Robert; Lattanze, Anthony J.; Stafford, Judith A.; Weinstock, Charles B.; & Wood, William G. "Quality Attribute Workshops", Third Edition (CMU/SEI-2003-TR- 016). Pittsburgh, PA.

[6] Clements, Paul C. "Active Reviews for Intermediate Designs" (CMU/SEI-2000-TN-009, ADA383775). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2000.

[7] Bass, Len; Clements, Paul; & Kazman, Rick. "Software Architecture in Practice", Second Edition. Boston, MA: Addison-Wesley, 2003.

[8] Kazman Rick "A Life-Cycle View of Architecture Analysis and Design Methods", September 2003, TECHNICAL NOTE, CMU/SEI-2003-TN-026.

[9] Keith Gallagher, Andrew Hatch and Malcolm Munro, "Software Architecture Visualization: An Evaluation Framework and Its Applications", IEEE Transactions on Software Engineering, Vol 34, No.2, March/April 2008.

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

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

[12] M. Storey, D.Cubranic,and D.German, "On the use of visualization to Support Awareness of Human Activities in software Development," Proc. ACM Symp.Software visualization,pp.193-202,2005.

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

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

[15] "IEEE Recommended Practice for Architectural Description of Software Intensive Systems," technical report, IEEE, 2000.

[16] 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.

[17] 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.
[18] B. Shneiderman, Designing the User Interface: Strategies for Effective Human-Computer Interaction. Addison-Wesley, 1998.
[19] 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.
[20] Sparx Systems, Enterprise Architect, http://www.sparxsystems. com.au, 2008.
[21] A. Rama Mohan Reddy, M.M. Naidu and P. Govindarajulu, "An Integrated approach of Analytical Hierarchy Process Model and Goal Model (AHP-GP Model) for selection of Software Architecture", IJCSNS International Journal of Computer Science and Network Security, Vol. 7, No.10, October 2007.
[22] K. Delhi Babu, P. Govindarajulu and A.N. Aruna Kumari, "Development of the Conceptual Tool for Complete Software Architecture Visualization: DArch (DA)", IJCSNS International Journal of Computer Science and Network Security, VOL.9 No.4, April 2009.
[23] L. Feijs and R. de Young, "3D Visualization of software architecture", Comm. ACM, Vol.41, No.12, pp.73-78, Dec, 1998.
[24] S. Basil and R. Keller, "A Qualitative and Quantitative Evaluation of Software Visualization", Proc. IEEE Workshop Visualization Software for Understanding and Analysis, pp.32-40, 2002.
[25] Keith Gallagher, Andrew Hatch and Malcolm Munro, "Software Architecture Visualization: An Evaluation Framework and Its Application", IEEE Transactions on Software Engineering, Vol.34, No.2, March/April 2008.

Authors

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

Dr. A. Ramamohana Reddy
Associate Professor
Department of Computer Science & Engineering
S.V. University, Tirupati,
Andhra Pradesh, India

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