Software-Architecture for Object Oriented Systems-Usability Patterns

R.V. Siva Balan[1], Dr. M. Punithavalli[2]

[1]Department of Computer Applications,
Narayanaguru College of Engineering,kanyakumari, India.

[2]The Director, Department of Computer Applications,
Dr. SNS Raja Lakshmi College of Arts and Science, Coimbatore, India.

Abstract Over the years the software engineering community has increasingly realized the important role of software architecture plays in fulfilling the quality requirements of a software system. It has been experienced that Software Architecture (SA) constrains the achievement of various quality attributes such as performance, security, maintainability and usability in a system. Reportedly, most software engineering projects reveals that a large number of usability related change requests are made after its deployment. Software patterns have proven to be a useful medium for capturing best practices. Building on the seminal work on software design patterns, usability patterns have been increasingly created to disseminate usability knowledge. In existing scenario-based software architecture analysis methods that focus on usability, the usage context is not employed to select scenarios used for analysis, it is known that understanding a specific usage context is important to carefully design for usability.

***Keywords**: Software Architecture, patterns, usability, quality attributes, QDK, design decision, ISO standards.*

1. Introduction

Since the last decades it has been clear that the most challenging activity of a software architect are not just designing for required domain functionality, but, also for specific quality attributes. One of the quality attribute which has its impact on users' acceptance is the usability. Usability has been disseminated as inherent to software quality because of the relationship between software and its application system domain. Issues such as whether a product is easy to learn to use and whether it is responsive to user determines its reputation. Usability engineering is the Human Computer Interaction (HCI) engineering and a discipline concerned with the design, evaluation and implementation of interactive computing systems for human use.

Adding usability improving solutions during late stage development is to some extent restricted by the software architecture. However, few software engineers and human–computer interaction engineers are aware of this important constraint and as a result avoidable rework is frequently necessary. User interface designers and software engineers have usually very different backgrounds resulting in the lack of mutual understanding of each other's view on technical or design issues.

Design patterns [14, 16] are extensively used by software engineers for the actual design process as well as for communicating a design to others. Since then a pattern community has emerged that specifies patterns for all sorts of problem domains: architectural styles [16], object-oriented frameworks [21], domain models of businesses [29], interaction patterns [34],[20],[31], etc. A lot of different types of patterns have been defined in the field of HCI; interaction patterns [34,20,31] (Undo), user interface patterns [38,30] (progress indicator), usability patterns [40,23] (Model view controller), web design patterns [30,10] (shopping cart) and workflow patterns [42].

The pattern idea was referenced by HCI research earlier than most people expect. Norman and Draper (1986) mention Alexander's work, and in his classic Psychology of Everyday Things [32], Norman states that he was influenced particularly by it. Apple's Human Interface Guidelines [1] quote Alexander's books as seminal in the field of environmental design, and the Utrecht School of Arts uses patterns as a basis for their interaction design curriculum [4].

Patterns have been shown to be a useful and potentially important vehicle for capturing some of the most significant architectural decisions [11]. One of the biggest difficulties of documenting architectural decisions is the capturing of rationale and expected consequences of a decision. This is where patterns are particularly strong, because the consequences of using the architecture pattern are part of the pattern. The result of applying a pattern is usually documented as "consequences" or "resulting context" and is generally labeled as positive ("benefits") or negative ("liabilities"). Each benefit and liability is described in some detail. The payoff of using patterns can be great. When an architect uses a pattern, he or she can read the pattern documentation to learn about the side effects of the pattern. This reduces the chance of the architect failing to consider important consequences. This relieves the architect of the burden of being expert in all the quality attributes. An important advantage of pattern-based architecting is that it is an integral part of most current architecture methods.

2. Software Architecture

Architecture is 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 IEEE 2000 standard [35] emphasizes that a system's software architecture is not only the model of the system at a certain point in time, but it also includes principles that guide its design and evolution. Software architecture is a key determinant of whether system quality requirements can be met. In software intensive systems, software architecture provides a powerful means to achieve the system qualities over the software life cycle [25]. Figure 1 shows how software architecture can help predict the quality of a software system. First, the relevant quality attributes are identified from the system's requirement specification. In the next step, these quality attributes are used to drive the software architecture which is subsequently used as a blueprint to assess the system's capabilities and qualities.

A number of case studies and theories based on practical experience have been published, suggesting the need for multiple architectural views to capture different aspects of software architecture [33]. The effectiveness of having multiple architectural views is that the multiple views help developers manage complexity of software systems by separating their different aspects into separate views [28]. Several architectural views share the fact that they address a static structure, a dynamic aspect, a physical layout and the development of the system. Bass et al. [25] introduced the concept of architecture structures as being synonyms to view.

Many software engineering textbooks describe the development stage between requirements and detailed design as architectural design and this is compatible with our notion of where the definition of the software architecture occurs, mapping the transition from problem definition to solution space. Whereas the ideas and motivations underlying software architecture are not novel, it is only within the past few years that researchers and practitioners have made explicit the architectural issues in their work, emphasizing the representation of the architecture as an important and living artifact in its own right within the life cycle of a system.

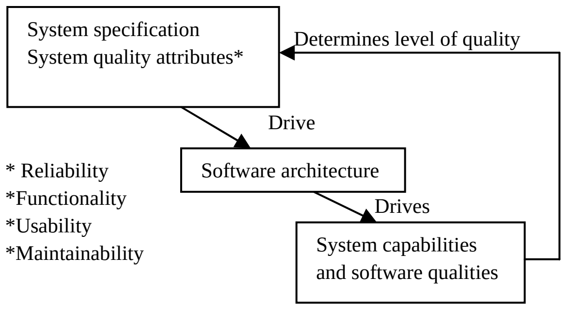

Fig 1: Relationship between software architecture and software quality.

Even more recent is the notion that an architectural representation may be analyzed to understand its fitness with respect to adapt quality attributes of the resulting system.

3. Design Patterns

Object-oriented programs evolve over time and it would be ideal if we could capture persistent parts of the programs early on and then derive the transient versions of the program from the persistent part. In our view, the object-oriented community is moving in this direction through its work on software architecture and patterns. Capturing the persistent parts of a program allows us to better maintain the integrity of the program during evolution. The most widely used concept of pattern in software development is the design pattern, and it is used particularly in the object-oriented paradigm. Design patterns reside in the domain of modules and interconnections. Design patterns can improve the structure of software, simplify maintenance, and help avoid architectural drift. Not all software patterns are design patterns. A variety of pattern categories are recognized in software pattern community. Note, nevertheless, that a design pattern can be seen as a unique or original solution. Design patterns have become an increasingly popular choice for addressing OOD’s limitations. Design patterns have a very close intact with the architectural design decisions. Architectural design decisions, among others, may be concerned with the application domain of the system, the architectural styles and patterns used in the system, COTS components and other infrastructure selections as well as other aspects needed to satisfy the system requirements. Abstracting the definition of design pattern, an architectural pattern can be defined as a description of the components of a design and the communication between these components to provide a solution for a usability pattern.

4. Quality Attributes

Industrial empirical studies reveals that Software Architecture constrains the achievement of various quality attributes (such as **performance, security, maintainability and usability**) in a system [5]. Since software architecture plays a significant role in achieving system wide quality attributes, it is important to analyze a system's software architecture with regard to desired quality requirements as early as possible [11]. The principle objective of software architecture analysis is to assess the potential of a proposed architecture to deliver a system capable of fulfilling required quality requirements and to identify any potential risks [27].

Quality attributes are characteristics of the system that are non-functional in nature. Because quality attributes are system-wide, their implementation must also be system-wide: satisfaction of a quality attribute requirement cannot be partitioned into a single module or subsystem. Thus, a system-level vision of the system is required in order to ensure that the system can satisfy its quality attributes. One of the primary purposes of the architecture of a system is to create a system design to satisfy the quality attributes. Architecture patterns are a viable approach for architectural partitioning, and have a well-understood impact on quality attributes [39]. However their application has been rather limited due to a number of factors [18].

Each domain has its own requirements for maintainability, performance, security, or usability. The requirement that there be a product line adds additional complexity to the design task but does not remove the necessity for designing to achieve all of the normal quality attributes for a domain. This work is a natural extension of the work of various communities. The patterns community believes that there are fundamental architectural patterns that underlie the design of most systems. Similarly, attribute communities ultimately explored the meaning of their particular attribute and come up with standard techniques for achieving their desired attribute.

5. Usability

The work in this paper is motivated by the fact that the pattern work also applies to usability. Usability is increasingly recognized as an important consideration during software development; however, many well-known software products suffer from usability issues that cannot be repaired without major changes to the software architecture of these products.

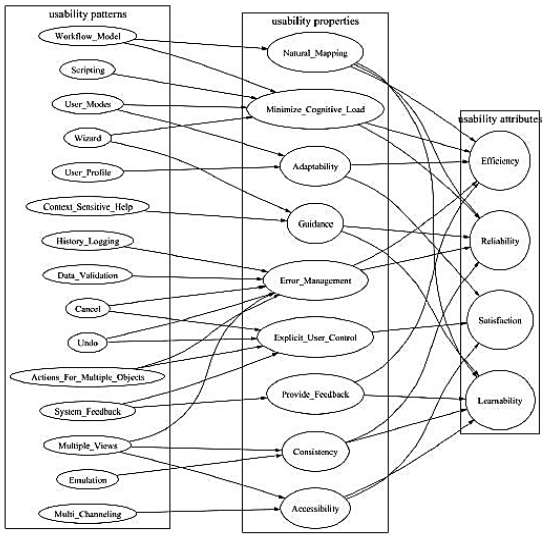

Fig 2: Usability Framework

The design and use of explicitly defined software architecture has received increasing amounts of attention during the last decade. Generally, three arguments for defining architecture are used [26]. First, it provides an artifact that allows discussion by the stakeholders very early in the design process. Second, it allows for early assessment of quality attributes [2, 9]. Finally, the design decisions captured in the software architecture can be transferred to other systems. This means that the design decisions embodied by software architecture are strongly influenced by the need to achieve quality attribute goals.

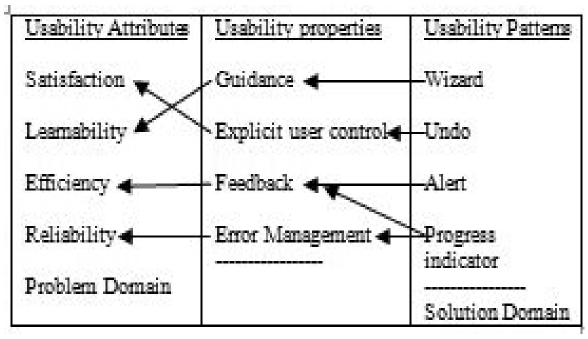

Fig 3: Relationship between usability patterns, properties and attributes.

Distributing usability knowledge is usability heuristics, which comes in many forms, from style guides (interface widgets) to general principles to interface consistency guidelines [36, 37]. While hundreds of usability guidelines have been designed and published, empirical studies have shown mixed results, with some demonstrating that both novice and expert HCI specialists benefit from guidelines [41, 15].

A combination of pattern maturity and new technologies have the potential to improve how patterns are used, created, and combined for form problem solutions. Use cases are a popular requirements modeling technique, yet people often struggle when writing them. They understand the basic concepts of use cases, but find that actually writing useful ones turns out to be harder than one would expect. One factor contributing to this difficulty is that we lack objective criteria to help judge their quality. Many people find it difficult to articulate the qualities of an effective use case. We have identified approximately three dozen patterns that people can use to evaluate their use cases. We have based these patterns on the observable signs of quality that successful projects tend to exhibit. Construction guidance is based on use case model knowledge and takes the form of rules which encapsulate knowledge about types of action dependency, relationships between actions and flow conditions, properties of objects and agents, etc. Based on this knowledge rules, help discovering incomplete expressions, missing elements, exceptional cases and episodes in the use case specification through pattern specification. They support the progressive integration of scenarios into a complete use case specification.

6. Usability in ISO 9126

In 1991, ISO 9126 defined usability as “a set of attributes that bear on the effort needed for use and on the individual assessment of such use, by a stated or implied set of users.” It then proposed a product-oriented usability approach. Usability was seen as an independent factor of software quality and it focused on software attributes, such as its interface, which make it easy to use [7]. In a product-oriented approach, usability is seen as a relatively independent contribution to software quality, as defined now in the 2001 edition of ISO/IEC 9126-1: “The capability of the software product to be understood, learned and liked by the user, when used under specified conditions.” Usable products can be designed by incorporating product characteristics and attributes, which are beneficial to users in particular contexts of use.

Usability specification and evaluation should address several user environments, which the software can affect, including both use preparation and results evaluation. To specify software quality, a purchaser needs a model and analytical tools to communicate

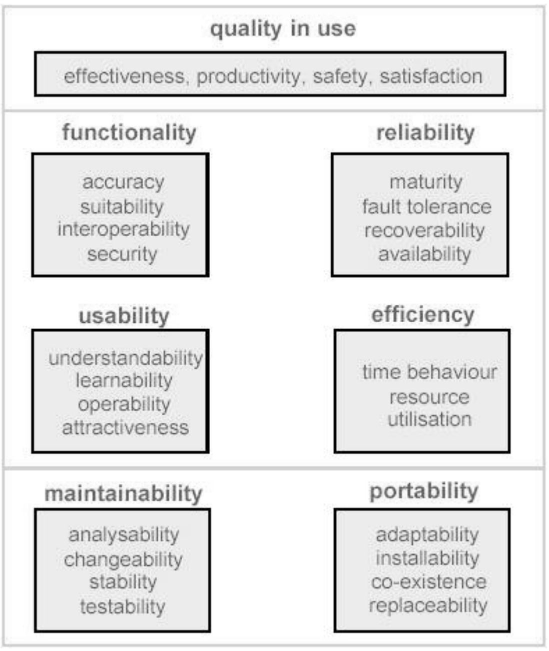

Fig 4: ISO 9126-1 Quality model

precisely his requirements concerning the product to be developed. Similarly, a software provider needs to be able to verify with confidence whether or not the product provides the expected level of software quality. This ISO 9126 standard can be used as a reference for contractual agreements between a purchaser and a software producer, and it can be used to eliminate a number of misunderstandings between purchaser and provider. The principal advantage of a clearly defined and agreed upon model, supported with appropriate measures, is that it clarifies the definition of usability, and proposes measures to provide objective evidence of achievement.

The ISO/IEC 9126 quality model can be used to specify and verify those properties that the software must exhibit before being put into service. However, there are still some weaknesses in ISO 9126 which have not yet been fully tackled, such as:

1. Unclear architecture at the detail level of the measures.
2. Some overlapping of concepts, making the standard challenging for the user community to grasp clearly, such as the usability characteristics of internal and external quality with respect to the quality-in-use set of quality characteristics.
3. Lack of a quality requirements standard.
4. Lack of guidance in assessing the results of measurement.

It is important to be able to relate software measures to project tracking and to target values at the time of delivery of the software.

7. Design Decisions and QDK

The process of architectural design has been characterized as making a series of decisions that have system-wide impact. Kruchten notes that the reasoning behind a decision is tacit knowledge, essential for the solution, but not documented [24]. The result is that consequences of decisions may be overlooked. Overlooking issues is a significant problem in architecture. In a study of architecture evaluations, Bass et al [6] report that most risks discovered during an evaluation arise from the lack of an activity, not from incorrect performance of an activity. Categories of risks are dominated by oversight, including overlooking consequences of decisions. Many of the overlooked consequences are associated with quality attributes. Their top risk themes included availability, performance, security, and modifiability. The iterative refinement of design decision (D_d), by means of the quality needs (Q_n) leads to the specification of (K_s) knowledge. This activity, QDK (Quality Needs to Design decision to Knowledge-rules Specification), explores the quality-impact design decision for usability. Most architectural decisions have multiple consequences; result in additional requirements to be satisfied by the architecture, which need to be addressed by additional decisions [22]. Some are intended, while others are side effects of the decision. Some of the most significant consequences of decisions are those that impact the quality attributes of the system. Garlan calls them key requirements [17]. We call it as Discovery of Knowledge, to be recorded in Knowledge rule Specification (K_s). This impact may be the intent of the decision; for example, one may choose to use a role-based access control model in order to satisfy a security quality attribute. Other impacts may be side effects of different decisions. For example, the architect may adopt a layered architecture approach, which decomposes the system into a hierarchy of partitions, each providing services to and consuming from its adjacent partitions. A side effect of a layered architecture is that security measures can be easily implemented. One of the key challenges in dealing with such consequences is the vast amount of knowledge required to understand their impact on all the quality attributes. Architectural design decisions are concerned with the application domain of the system, the architectural styles and patterns used in the system, COTS components and other infrastructure selections as well as other aspects needed to satisfy the system requirements.

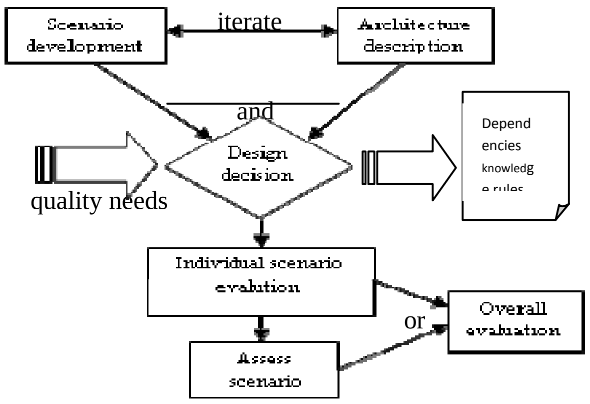

Fig 5: Activities and dependencies in QDK scenario based analysis

Bachmann et al note that the list of quality attributes in the ISO 9126 standard is incomplete, and that one must understand the impact on even the undocumented quality attributes [3]. Missing the impact on quality attributes at architecture time has an additional liability. Because quality attributes are system-wide capabilities, they generally cannot be fully tested until system testing [8]. Consequences that are overlooked are often not found until this time, and are expensive to fix.

8. Result and Future work

When the scenario evaluation has been finished we need to interpret the results to draw our conclusions concerning the software architecture. At this stage we go back to our architecture design approach where we wondered if this architecture had sufficient support for usability. The interpretation of the results depends entirely on the goal of the analysis and the system requirements. If the architecture proves to have sufficient support for all quality attributes the design process is ended. Otherwise we need to apply architecture transformations or design decisions to improve certain quality attribute(s). The choice to use particular transformations may be based upon results from the analysis. For example: Consider a system, which proves to have a low support for usability, for example learnability for some usage scenarios is not supported. To improve learnability we could use the design primitive of guidance, to address guidance we could implement for example a wizard pattern or provide context sensitive help. The framework we have developed is then used as an informative source for design and improvement of usability. Several issues need to be resolved during case studies, which have been summarized below:

i. Relevance of framework: The relationships depicted in our framework indicate potential relationships.

Further work is required to substantiate these relationships.

ii. Use case maps: may provide information about static properties of usability. More research is required to determine whether use case maps can provide that kind of information.

9. Conclusions

The work presented in this paper is motivated by the increasing realization in the software engineering community of the importance of software architecture for fulfilling quality requirements. We have presented a provisional assessment technique for usability based on scenarios, which has potential to improve current design for usability. Future case studies should determine the validity of our approach to refine it, possibly redefine and elaborate the steps that should be taken to make it generally applicable. The main contribution of this paper is the formulation and derivation of an architectural assessment approach for usability.

References

[1] Apple Computer (I992), Macintosh Human Interface Guidelines. Addison-Wesley, Reading, MA.

[2] Architecting for usability; a survey, http://segroup.cs.rug.nl.

[3] Bachmann, F., Bass, L., Klein, M., Shelton, C.: Designing software architectures to achieve quality attribute requirements. In: IEEE Proceedings, vol. 152 (2005).

[4] Barfield, L., van Burgsteden, W., Lanfermeijer, R. et al. (1994). Interaction Design at the Utrecht School of the Arts, SIGCHI Bulletin. 26(3). 49-79.

[5] BASS, L., CLEMENTS, P. & KAZMAN, R. (2003) Software Architecture in Practice, Addison-Wesley.

[6] Bass, L., Nord, R., Wood, W., Zubrow, D.: Risk Themes Discovered Through Architecture Evaluations. SEI Report CMU/SEI-2006-TR-012 (2006).

[7] Bevan, N. 1997. ISO 9126, EAGLES evaluation group workshop, Evaluation in Natural Language Engineering:Standards and Sharing, Brussels, November 26th and 27th, http://www.cst.ku.dk/projects/eagles2/workshop/ISOnigel.html.

[8] Burnstein, I.: Practical Software Testing. Springer, Heidelberg (2003).

[9] D. Kim, R. France, S. Ghosh and E. Song, “A UMLBased Metamodeling Language to Specify Design Patterns”, Proceedings of Workshop on Software Model Engineering (WiSME), at UML 2003, San Francisco, 2003.

[10] D.K. van Duyne, J.A. Landay, J.I. Hong, The Design of Sites: Patterns, Principles, and Processes for Crafting a Customer-Centered Web Experience, Addison-Wesley, Boston, 2002.

[11] DOBRICA, L. & NIEMELA, E. (2002) A Survey on Software Architecture Analysis Methods. IEEE Transactions on Software Engineering, 28, 638-653.

[12] E. Folmer et al. / Information and Software Technology 48 (2006).

[13] E. Folmer, J. v. Gurp, and J. Bosch. Scenario-Based Assessment of Software Architecture Usability. In the Proceedings of Workshop on Bridging the Gaps Between Software Engineering and Human-Computer Interaction, ICSE, 2003.

[14] E. Gamma, R. Helm, R. Johnson, J. Vlissides, Design Patterns Elements of Reusable Object-Orientated Software, Addison-Wesley, Reading, MA, 1995.

[15] E. Lai-Chong Law, E. T. Hvannberg, "Analysis of Strategies for Improving and Estimating the Effectiveness of Heuristic Evaluation," Proc. 3rd Nordic Conf. on HCI, pp. 241-250, 2004.

[16] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, M. Stal, Pattern-Oriented Software Architecture: A System of Patterns, Wiley, New York, 1996.

[17] Garlan, D.: Software Architecture: a Roadmap. In: Proceedings of Future of Software Engineering, Limerick Ireland (2000).

[18] Harrison, N., Avgeriou, P., Zdun, U.: Architecture Patterns as Mechanisms for Capturing Architectural Decisions. IEEE Software (September/October 2007).

[19] J. K. Bergey, M. J. Fisher and L. G. Jones and R. Kazman. Software Architecture Evaluation with ATAMSM in the DoD System Acquisition Context. CMU/SEI- 99-TN-012. Pittsburg, PA: Software Engineering Institute, Carnegie Mellon University, 1999.

[20] J. Tidwell, Interaction design patterns, Proceedings of the Conference on Pattern Languages of Programming 1998, Illinois USA, 1998.

[21] J.O. Coplien, D.C. Schmidt, Pattern Languages of Program Design, Software Patterns Series, Addison-Wesley, Reading, MA, 1995.

[22] Jansen, A.G., Bosch, J.: Software Architecture as a set of Architectural Design Decisions. In: Proceedings of WICSA 5, pp. 109–119 (November 2005).

[23] K. Perzel, D. Kane, Usability patterns for applications on the world wide web, Proceedings of the Pattern Languages of Programming Conference, 1999.

[24] Kruchten, P., Lago, P., van Vliet, H.: Building up and reasoning about architectural knowledge. In: Hofmeister, C., Crnkovic, I., Reussner, R. (eds.) QoSA 2006. LNCS, vol. 4214, Springer, Heidelberg (2006).

[25] L. Bass, P. Clements and R. K. Kazman. Software Architecture in Practice. SEI Series in Software Engineering. Addison-Wesley, 1998. ISBN 0-201-19930-0.

[26] L. Bass, P. Clements, and R. Kazman, Software Architecture in Practice, Addison Wesley Longman, Reading MA, 1998.

[27] LASSING, N., RIJSENBRIJ, D. & VLIET, H. V. (1999) The goal of software architecture analysis: Confidence building or risk assessment. Proceedings of First BeNeLux conference on software architecture.

[28] M. A. Babar and I. Gorton. Comparison of Scenario-Based Software Architecture Evaluation Methods. In the Proceedings on Asia-Pacific Software Engineering Conference, pp. 584-585, 2004.

[29] M. Fowler, Analysis Patterns: Reusable Object Models, Addison-Wesley, Reading, MA, 1996.

[30] M. Welie, GUI Design Patterns, http://www.welie.com/
[31] M. Welie, H. Traetteberg, Interaction patterns in user interfaces, Proceedings of the Seventh Conference on Pattern Languages of Programming (PloP), 2000.
[32] Norman, D.A. (1988). The Psychology of Everyday Things. Basic Books, New York.
[33] P. Clements and R. K. Kazman, M. Klein. Evaluating Software Architectures: Methods and Case Studies. Addison-Wesley Professional; 2002. ISBN 0-201-70482X.
[34] Pointer, PoInter: Patterns of INTERaction collection,http://www.comp.lancs.ac.uk/computing/research/cseg/projects/pointer/patterns.html.
[35] Recommended practice for architectural description. IEEE Standard P1471, 2000.
[36] S. J. Koyanl, R. W. Bailey, J. R. Nall, "Research-Based Web Design & Usability Guidelines," Communication Technology Branch, National Cancer Institute & US Dept of health and Human Services,http://www.usability.gov/pdfs/guidelines.html, 2003.
[37] S. L. Smith, J. N. Mosier, "Guidelines for Designing User Interface Software," ESD-TR-86-278, Technical Report, The MITRE Corporation, 1986.
[38] S.A. Laakso, User Interface Design Patterns, http://www.cs.helsinki.fi/u/salaakso/patterns/
[39] Shaw, M., Garlan, D.: Software Architecture: Perspectives on an Emerging Discipline. Addison-Wesley, Reading, MA (1996).
[40] UK, The Usability Group at the University of Brighton, The Brighton Usability Pattern Collection,http://www.cmis.brighton.ac.uk/research/patterns/home.html.
[41] W. Connell, N. V. Hammond, "Comparing usability evaluation principles with heuristics," Proc. INTERACT, pp. 621-636, 1999.
[42] W. van der Aalst, A. ter Hofstede, Workflow Patterns,http://tmitwww.tm.tue.nl/research/patterns/

Prof. Dr. M. Punithavalli is currently the Director, Department of Computer Applications, Dr. SNS College of Arts and Science, Coimbatore, India. She is actively working as the Professor in the department of Computer Applications of SNS Raja Lakshmi College of Engineering, India.

Lect. R.V. Siva Balan is currently working as the Lecturer in the Department of Computer Applications, Narayanaguru College of Engineering, Kanyakumari Dist., India. He is a research scholar in Anna University Coimbatore, India.