Journal of Software Engineering

ISSN 1819-4311

Journal of Software Engineering 5 (3): 102-107, 2011
ISSN 1819-4311 / DOI: 10.3923/jse.2011.102.107
© 2011 Academic Journals Inc.

Practice Based Guidelines for Effective Software Architecture of Web Based Applications

Parminder Kaur and Hardeep Singh
Department of Computer Science and Engineering, Guru Nanak Dev University, Amritsar-143005, India

Corresponding Author: Parminder Kaur, Department of Computer Science and Engineering, Guru Nanak Dev University, Amritsar-143005, India

ABSTRACT
Software architecture is emerging as an important discipline for engineers of software. Software architects have been limited by a lack of standardized ways to represent architecture as well as analysis methods to predict whether an architecture will result in an implementation that meets the requirements. Architects also have had little guidance in how to go about designing the architecture, which decisions should be made first what level of detail the architecture should encompass, how conflicting concerns should be satisfied and what range of issues the architecture should cover. This study makes an attempt to illustrate architectural design guidance in form of functional dimensions and structural dimensions required to identify the requirements as well as overall structure of a user-interface system.

Key words: Software architecture, architectural design, design space dimensions, functional dimensions, structural dimensions, architectural styles, architectural patterns

INTRODUCTION
Software architecture is emerging as a natural evolution of design abstractions for engineering the software. The success of a software system depends on a good architectural design. As the size and complexity of software systems increase, the design and specification of overall system structure become more significant issues than the choice of algorithms and data structures for computation. Software architectural patterns and styles deal with various structural issues like organization of a system as a composition of components, global control structures, the protocols for communication, synchronization and data access, the assignment of functionality to design elements, the composition of design elements, physical distribution, scaling and performance, dimensions of evolution and selection among design alternatives (Taylor *et al*., 2009; Garlan and Shaw, 1994, 2010).

Architectural design guidance helps in formulating design rules that indicate good and bad combinations of choices and use them to select an appropriate system design based on functional as well as structural dimensions (Lane, 1990a, b).

Software architecture encompasses the structures of large software systems, where every system comprises of elements and the relations among them. "The software architecture of a program or computing system is the structure or structures of the system, which comprise software

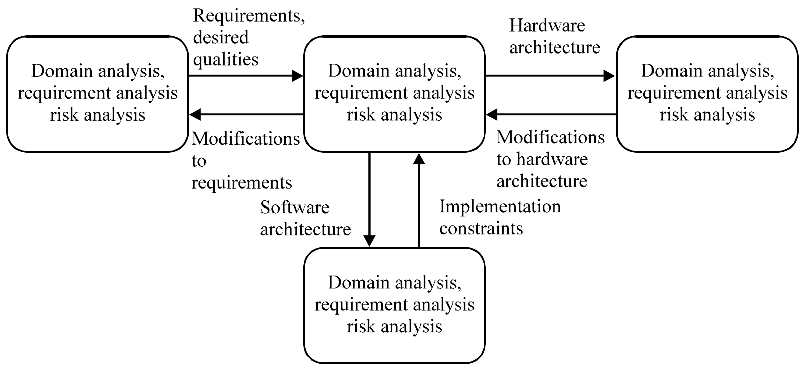
Requirements, desired qualities
Hardware architecture
Domain analysis, requirement analysis risk analysis
Domain analysis, requirement analysis risk analysis
Domain analysis, requirement analysis risk analysis
Modifications to requirements
Modifications to hardware architecture
Software architecture
Implementation constraints
Domain analysis, requirement analysis risk analysis

Fig. 1: Relation of software architecture to other development activities (Hofmeister *et al.*, 2000)

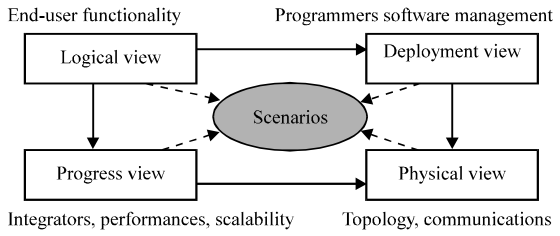

Fig. 2: The '4+1' view model (Muskens, 2002)

elements, the externally visible properties of those elements and the relationships among them, (Bass *et al.,* 2003). According to Boehm (1995), If a project has not achieved a system architecture, including its rationale, the project should not proceed to full-scale system development. Specifying the architecture as a deliverable enables its use throughout the development and maintenance process. Jacobson *et al.* (1999) defines software architecture as "An architecture is the set of significant decisions about the organization of a software system, the selection of the structural elements and their interfaces by which the system is composed, together with their behavior as specified in the collaborations among those elements, the composition of these structural and behavioral elements into progressively larger subsystems and the architectural style that guides this organization these elements and their interfaces, their collaboration and their composition". Fig. 1 shows that how software architecture fits in with other development tasks.

As its essence, software architecture is defined as a set of principal design decisions made about the system. The '4+1' view model is depicted in Fig. 2 (Muskens, 2002), which consists of logical view, development view, process view and physical view along with use cases or scenarios which can be considered as fifth view. For each view, most of the design decisions are independent of other views but there are some decisions that are affected by the views that are designed later.

Few common software architectures: The success of a software system depends on a good architectural design. There are a number of common software architectural styles and patterns such as pipelines, client-server organization, layered architecture, component-based architecture, message bus architecture and Service-Oriented Architecture (SOA) (Garlan *et al.*, 1992;

Table 1: Common software architectures

Category	Architecture styles
Communication	Service-Oriented Architecture (SOA), message bus, pipes and filters, event-based, implicit invocation
Deployment	Client/Server, N-Tier, 3-Tier
Domain	Domain driven design
Data-centered	Repositories
Structure	Component-based, object-oriented, layered architecture
Virtual machines	Interpreters

Shaw, 1990, 1991, 1993; Allen and Garlan, 1992; Erich *et al.*, 1995; Pree, 1995). Table 1 lists the major areas of focus and the corresponding software architectures (Garlan and Shaw, 1994).

Design space dimensions: The notion of design space is useful in its own right as a shared vocabulary for describing and understanding systems for specific domains. A multidimensional design space helps in classifying system architectures. Each dimension describes variation in one system characteristics or design choice. A specific system design corresponds to a point in the design space, identified by the dimensional values that describe its characteristics and structure. Lane (1990a) discussed two major types of dimensions i.e., functional dimensions and structural dimensions. Functional dimensions identify the requirements for user-interface system that most affect its structure. It deals with the requirements of particular applications, users, I/O devices to be supported, constraints imposed by the surrounding computer system, key decisions about the user-interface behavior, development cost considerations and degree of adaptability of the system. Structural dimensions deals with the decisions that determine the overall structure of a user-interface system. It deals with the issues like how system functions are divided into modules, the interfaces between modules, information contained within each module, data representations used within the system and dynamic behavior of the user-interface code.

In order to study the practices followed by the undergraduate students with respect to the various architectural features for the development of different web-based applications. Two primary dimensions of software architecture namely functional dimensions and structural dimensions were considered as the benchmarks for evaluating these projects. A total of 30-35 projects with respect to different types of applications, were taken into consideration as part of the study. The various features available in these projects are listed in the Table 2 and Table 3 against the threshold features.

It has been conclusively found that the majority of the projects make use of at the most six to seven features of architectural dimensions. This can be attributed to a number of reasons namely:

- The lack of awareness of various architectural practices among the students
- International ignorance of various architectural features because of paucity of time and resources. Thereby compromising on the overall application quality
- Lack of the specification of exact requirements with respect to the architecture of the overall application
- The ad-hoc development approaches result in degradation of overall software architecture over a period of time

Table 2: Functional dimensions for web-based user-interface systems

Projects/functional dimensions	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	P11	P11	P13	P14
External event handling														
• No external events			✔	✔	✔	✔	✔				✔	✔	✔	
• Process events while waiting for input	✔							✔	✔	✔				✔
• External events preempt user commands		✔												
User customizability														
• High			✔	✔										
• Medium	✔	✔			✔			✔	✔		✔	✔	✔	✔
• Low						✔	✔			✔				
User interface adaptability across devices														
• None		✔	✔	✔	✔	✔	✔			✔	✔	✔	✔	
• Local behavior changes	✔							✔	✔					✔
• Global behavior changes														
• Application semantics changes														
Computer system organization														
• Uniprocessing														
• Multiprocessing	✔	✔	✔		✔			✔		✔			✔	
• Distributed processing				✔		✔	✔		✔		✔	✔		✔
Basic interface class														
• Menu selection	✔		✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	
• Form filling	✔	✔	✔			✔	✔	✔	✔	✔	✔	✔	✔	✔
• Command language			✔											
• Natural language									✔		✔	✔		
• Direct manipulation	✔							✔						
Application portability across user interface styles														
• High														
• Medium	✔	✔		✔	✔	✔	✔	✔	✔	✔	✔	✔		✔
• Low														

Table 3: Structural dimensions for web-based user-interface systems

Projects/structural dimensions	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	P11	P12	P13	P14
Application interface abstraction level														
• Monolithic progra														
• Abstract device												✔		
• Toolkit	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔		✔	✔
• Interaction manager with fixed data types														
• Interaction manager with extensible data types														
• Extensible Interaction manager														
Abstract device variability														
• Ideal device		✔										✔		
• Parameterized device			✔	✔	✔	✔		✔	✔	✔	✔		✔	✔
• Device with variable operations														
• Ad-hoc device	✔							✔						
Notation for user interface definitions														
• Implicit in shared user interface code	✔							✔						✔
• Implicit in application code		✔												✔
• External declarative notation												✔		
• External procedural notation		✔			✔		✔			✔	✔	✔	✔	✔

Table 3: Continued

Projects/structural dimensions	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	P11	P12	P13	P14
• Internal declarative notation														
• Internal procedural notation				✔	✔	✔	✔				✔	✔		✔
Basis of communication														
• Events			✔	✔	✔	✔	✔			✔	✔	✔	✔	✔
• Pure state														
• State with hints	✔							✔						
• State plus events		✔							✔					
Control thread mechanism														
• None	✔							✔						
• Standard processes		✔										✔		
• Lightweight processes														
• Non-preemptive processes														
• Event handlers			✔	✔	✔	✔	✔		✔	✔	✔	✔	✔	
• Interrupt service routines														

CONCLUSIONS AND FUTURE WORK

In order to develop a good quality software based on a strong architecture it is suggested that the corresponding practices be exhaustively taught to the developers of various applications. Their effective implementations must also be effectively demonstrated. This shall remove the gap between the acquired levels of competence in this area and desired levels at the application development level. However, this should all be the part of overall application development environment which includes different processes like requirement engineering and the translation of the software architecture into effective design and code.

There is a requirement to supplement these results with the study of applications from other domains like database applications, file-based applications and the function-based applications.

ACKNOWLEDGMENT

We thankfully acknowledge the contribution of various students of our department towards this study.

REFERENCES

Allen, R. and D. Garlan, 1992. A formal approach to software architectures. Proceedings of the IFIP 12th World Computer Congress, September 7-11, 1992, Elsevier Science Publishers, Madrid, Spain.

Bass, L., P. Clements and R. Kazman, 2003. Software Architecture in Practice. 2nd Edn., Addison-Wesley Publishing Co., Boston, Massachusetts, ISBN: 0-321-15495-9.

Boehm, B., 1995. Engineering context. Proceedings of the 1st International Workshop on Architectures for Software Systems, April 24-25, 1995, Seattle, Washington.

Erich, G., H. Richard, J. Ralph and V. John, 1995. Design Patterns: Elements of Reusable Object-Oriented Design. Addison Wesley, Reading, MA, USA.

Garlan, D., M. Shaw, C. Okasaki, C. Scott and R. Swonger, 1992. Experience with a course on architectures for software systems. Proceedings of the 6th SEI Conference on Software Engineering Education, October 5-7, 1992, Springer-Verlag.

Garlan, D. and M. Shaw, 1994. An Introduction to Software Architecture. Vol. 94-166, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA.

Garlan, D. and M. Shaw, 2010. Software Architecture, Perspectives on an Emerging Discipline. PHI Learning Pvt. Ltd., New Delhi.

Hofmeister, C., R. Nord and D. Soni, 2000. Applied Software Architecture. Addison-Wesley Professional, Reading, MA, USA, ISBN: 9780201325713, Pages: 397.

Jacobson, I., G. Booch and J. Rambaugh, 1999. The Unified Software Development Process. 1st Edn., Addison-Wesley Longman, MA., USA., ISBN-10: 0-201-57169, Pages: 512.

Lane, T.G., 1990a. Studying software architecture through design spaces and rules. Technical Report, CMU/SEI-90-TR-18, ESD-90-TR-219, Software Architecture Design Principles Project, Carnegie Mellon University, http://www.sei.cmu.edu/reports/90tr018.pdf.

Lane, T.G., 1990b. User interface software structures. Ph.D. Thesis, Carnegie Mellon University.

Muskens, J., 2002. Software architecture analysis tool. Master Thesis, Technische Universiteit Eindhoven, Department of Mathematics and Computing Science.

Pree, W., 1995. Design Patterns for Object-Oriented Software Development. Addison-Wesley Pub. Co., Reading, Massachusetts, ISBN: 9780201422948, Pages: 268.

Shaw, M., 1990. Toward higher-level abstractions for software systems. Data Knowledge Eng., 5: 119-128.

Shaw, M., 1991. Heterogeneous design idioms for software architecture. Software Proceedings of the Sixth International Workshop on Specification and Design, October 25-26, 1991, Como, Italy, pp: 158-165.

Shaw, M., 1993. Software Architectures for Shared Information Systems. In: Mind Matters: Contributions to Cognitive and Computer Science in Honor of Allen Newell, Steier, D. and T. Mitchell (Eds.). Erlbaum, Hillsdale, NJ.

Taylor, R.N., N. Medvidovic and E.M. Dashofy, 2009. Software Architecture: Foundations, Theory and Practice. John Wiley and Sons, Chichester, ISBN: 9780470167748, Pages: 712.