A Case Study Based Approach for Effective Use of Software Architecture Guidelines

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

Hardeep Singh
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. A case study is performed to illustrate architectural design guidance in form of functional dimensions and structural dimensions essential to identify the requirements as well as overall structure of database systems.

General Terms
Software Architecture, Architectural Design, Architectural Styles, Architectural Patterns

Keywords
Design Space Dimensions, Functional Dimensions, Structural Dimensions

1. 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, 2009; Garlan, and Shaw, 1994; 2010).

The software architecture should define and describe the elements of the system at a relatively coarse granularity. It should describe how the elements fulfill the system requirements, including which elements are responsible for which functionality, how they interact with each other, how they interact with the outside world and their dependencies on the execution platform. 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 (Thomas, 1990 (a & b)).

2. SOFTWARE ARCHITECTURE TERMINOLOGY
Software architecture encompasses the structures of large software systems, where every system comprises of elements and the relations among them. Table 1 shows the comparison of some of the terms used to describe software architectures.

Table 1: Comparison of Software Architecture Terms [Booch *et al*, 1999]

Term	Define Element Types and How They Interact	Define a Mapping of Functionality to Architecture Elements	Define Instances of Architecture Elements
An **Architectural Style** or **Architectural Pattern** (Usually not domain specific)	Yes	Sometimes	No
A **reference architecture** or **domain - specific software architecture** (applies to a particular domain)	Yes	Yes	No
A **product-line architecture** (applies to a set of products within an organization)	Yes	Yes	Sometimes
A **software architecture** (applies to a system or product)	Yes	Yes	Yes

3. SOFTWARE ARCHITECTURE DEFINITIONS
According to (Bass et. al, 2003), "The software architecture of a program or computing system is the structure or structures of the system, which comprise software elements, the

externally visible properties of those elements and the relationships among them". (Boehm, 1995) explains, "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". (Booch, Rumbaugh, and Jacobson, 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 collaborations, and their composition". Figure 1 shows that how software architecture fits in with other development tasks.

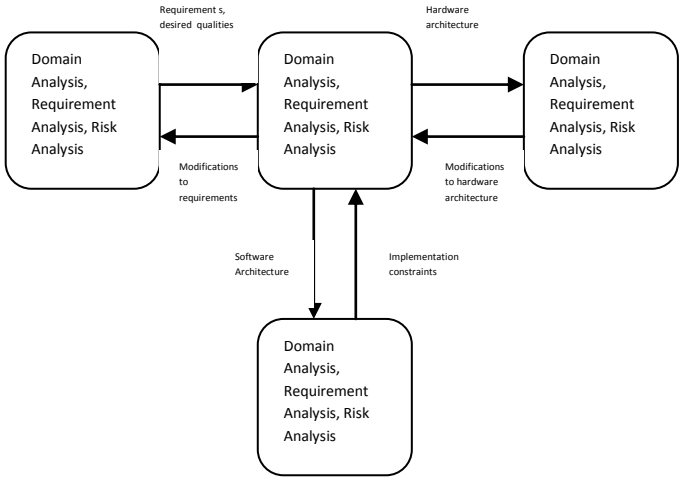

Fig. 1: Relation of software architecture to other development activities [Hofmeister *et al* (2000)]

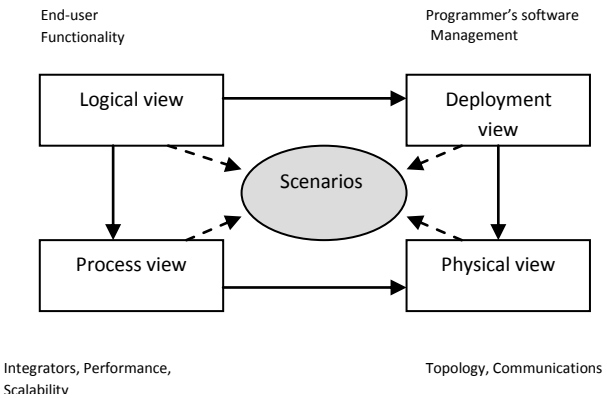

Fig. 2: The '4+1' view model [Muskens, 2002]

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 figure 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.

4. 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; Shaw, 1990, 1991, 1993, 1994, 2010; Allen and Garlan, 1992, Erich *et.* al 1995; Wolfgang, 1995). Table 2 lists the major areas of focus and the corresponding software architectures (Garlan, and Shaw, 1994, Gorton, 2006).

Table 2: 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

5. 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. Thomas, (1990) 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 database 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 15 - 20 projects with respect to different types of database management systems, were taken into consideration as part of the study. The various features available in these projects are listed in the table 3 and table 4 against the threshold features.

Table 3: Functional Dimensions for Database Systems

Projects (Database) / **Functional Dimensions**	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	P11	P12	P13	P14	P15	P16
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 4: Structural Dimensions for Database Systems

Projects (Database) / **Structual Dimensions**	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	P11	P12	P13	P14	P15	P16
Application Interface abstraction level																
• Monolithic program																
• 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	✓						✓		✓							
• 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																

The major prominently observed features are:

- Lack of sight for the global behavior of the User Interface.
- There is no consideration for semantic changes.
- User Customization is neglected.
- Application portability across user interface is negligible.
- No use of non-preemptive processes.
- Use of Interrupt service routines is negligible.

It has been conclusively found that the majority of the projects make use of very few 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.
- Intentional 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.

6. 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.

This is not to say that more work is not needed in this field (Garlan, and Shaw, 1994). There is a need to expect significant advances in a number of areas including better software architecture taxonomies, better taxonomies of architectural styles, formal models for characterizing and analyzing architectures, better understanding of the primitive semantic entities from which these styles are composed, enhanced notations for describing architectural designs, better tools and environments for developing architectural designs, improved techniques for extracting architectural information from existing code and better understanding of the role of architectures in the life-cycle process.

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

7. ACKNOWLEDGMENT

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

8. REFERENCES

[1] Active Reviews for Intermediate Design (ARID), available at http://www.sei.cmu.edu/architecture/tools/evaluate/arid.cfm

[2] Allen, R. and Garlan, D. (1992), "A formal approach to software architectures," in *Proceedings of IFIP'92* (J. van Leeuwen, ed.), Elsevier Science Publishers B.V.

[3] Architecture Tradeoff Analysis Method (ATAM), available at http://www.sei.cmu.edu/architecture/tools/evaluate/atam.cfm

[4] Bass, L; Clements, P. and Kazman, R. (2003), Software Architecture in Practice, 2nd Edition, Addison Wesley.

[5] Boehm, B. (1995), "Engineering Context", *Proceedings of the First International Workshop on Architectures for Software Systems*. Available on CMU-CS-TR-95-151 from the school of Computer Science, Carnegie Mellon University.

[6] Booch, G,; Rumbaugh, J. and Jacobson, I. (1999), *The Unified Software Development Process*. Addison-Wesley Professional, ISBN 0-201-57169-2.

[7] Cost Benefit Analysis Method (CBAM), available at http://www.sei.cmu.edu/architecture/tools/evaluate/cbam.cfm

[8] Erich, G.; Richard, H.; Ralph, J. and John, V. (1995), *Design Patterns: Elements of Reusable Object-Oriented Design*, Addison Wesley.

[9] Garlan, D. and Shaw, M. (1994), An Introduction to Software Architecture, CMU-CS-94-166, see at http://www.cs.cmu.edu/afs/cs/project/able/ftp/intro_softarch/intro_softarch.pdf

[10] Garlan, D. and Shaw, M. (2010), *Software Architecture, Perspectives On An Emerging Discipline*, PHI Learning

[11] Garlan, D.; Shaw, M.; Okasaki, C.; Scott, C. and Swonger, R. (1992), "Experience with a course on architectures for software systems," in *Proceedings of the Sixth SEI Conference on Software Engineering Education*, Springer-Verlag, LNCS 376.

[12] Homeister C., Nord R. and Soni D. (2000), *Applied Software Architecture*, Addison Wesley

[13] Muskens, J. (2002), *Software Architecture Analysis Tool*, Master Thesis, TECHNISCHE UNIVERSITEIT EINDHOVEN, Department of Mathematics and Computing Science, available at: http://www.win.tue.nl/~clange/empanada/thesis_johanmuskens%20v01a.pdf

[14] Shaw, M. (1990), "Toward higher-level abstractions for software systems," in *Data & Knowledge Engineering*, vol. 5, pp. 119-128, North Holland: Elsevier Science Publishers B.V.

[15] Shaw, M. (1991), "Heterogeneous design idioms for software architecture," in *Proceedings of the Sixth International Workshop on Software Specification and Design, IEEE Computer Society, Software Engineering Notes*, (Como, Italy), pp. 158-165.

[16] Shaw, M. (1993), "Software architectures for shared information systems," in *Mind Matters: Contributions to Cognitive and Computer Science in Honor of Allen Newell*, Erlbaum.

[17] System and Software ATAM, available at http://www.sei.cmu.edu/architecture/tools/evaluate/systematam.cfm

[18] Taylor, R.N. (2009), *Software Architecture: Foundations, Theory, and Practice*, Wiley Publications, ISBN-10:0470167742, ISBN-13:9780470167748.

[19] Thomas G. Lane (1990a), "Studying software architecture through design spaces and rules", Technical Report CMU/SEI-90-TR-18 ESD-90-TR-219 and CMU-CS-90-175, Carnegie Mellon University.

[20] Thomas G. Lane (1990b), *User Interface Software Structures*, Ph.D. thesis, Carnegie Mellon University.

[21] Wolfgang, P. (1995), *Design Patterns for Object-Oriented Software Development.* Addison Wesley, ACM Press.