Evaluation and Classifying Software Architecture Styles Due to Quality Attributes

Nassrin Eftekhari, Morteza Poyan Rad, Hamid Alinejad-Rokny

1Electrical & Computer Engineering Faculty, Qazvin Islamic Azad University, Qazvin, Iran.
2Electrical & Computer Engineering Faculty, Qazvin Islamic Azad University, Qazvin, Iran.
3Department of Computer Engineering, Science and Research Branch, Islamic Azad University, Tehran, Iran.

Abstract: One of the important concepts in software architecture evaluation is measurement of software quality attributes. From 1970 the Software Engineering Institutes pay attention to some of these quality attributes such as performance, security, availability, reusability and so on. In this paper we try to show most of quality attributes for all kind of software architecture styles and then summarized its goals, properties and related quality attributes. Then all kind of quality attributes and metrics of software were investigated. Finally we introduce a framework for all quality attributes in each style and present samples of each architecture styles. Regarding to performed evaluation in this paper we will propose that what kind of architecture styles is suitable for which applications.

Key words: Software architecture, Software architecture style, System quality attributes, software architecture evaluation, Software quality measurement.

INTRODUCTION

Software architecture is being widely used today to describe a very high level design methodology of large software systems. Software architecture represents the overall structure of a system in an abstract, structured manner.

The architecture deals with the structure of the components of a system, their interrelationships and guidelines governing their design and evolution over time. So, a description of architecture is composed of identifiable components of various distinct types (Babar and Gorton, 2009).

The components comprising architecture could include objects, clients and servers, databases, filters and layers. These components could interact using several mechanisms such as procedure calls, message sending, shared variable access, clients /server protocols.

The general purpose software architecture can be defined in terms of its components, component interactions, constraints, and control structure (Williams and Jeffrey, 2010). Some commonly occurring patterns of the structural organization of components and connectors have been identified, and these patterns are referred to as architecture styles. An architecture style defines a family of software systems in terms of a pattern of software organization.

The software architecture conforms to some style. Therefore, since every software system has the architecture, every software system has a style, and styles must have existed since the first software system was developed.

In the next section, we describe different software architecture definitions. In section 3 we present software quality attributes. Section 4 details software architecture styles and quality attributes related to each Style. Finally, section 5 concludes the paper.

2. Software Architecture:

Software architectures have been proposed to address the challenges of growing complexity and size in modern, distributed software systems.

Software architectures provide high-level abstractions in the form of coarse-grained processing, connecting, and data elements, their interfaces, and their configurations (Garlan and Perry, 1995). This additional level of abstraction, while aiding comprehension and construction of software systems, and also requires additional effort for its use (Mehta and Medvidovic, 2003; Taylor et al., 2009).

There exist a number of definitions of software architecture with minor differences depending on domain and people’s experience. However, most definitions share common characteristics. Table 1 provides different definitions that have been presented for software architecture.
Table 1: Different definitions of software architecture.

<table>
<thead>
<tr>
<th>Reference</th>
<th>Definition</th>
<th>Year</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Bass &amp; al, 2003)</td>
<td>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.</td>
<td>2003</td>
</tr>
<tr>
<td>(Perry &amp; Wolf, 1992)</td>
<td>A set of architectural (or, if you will, design) elements that have a particular form. Perry and Wolf distinguish between processing elements, data elements, and connecting elements, and this taxonomy by and large persists through most other definitions and approaches.</td>
<td>1992</td>
</tr>
<tr>
<td>(Garlan &amp; Shaw, 1993)</td>
<td>Software architecture is a level of design concerned with issues beyond the algorithms and data structures of the computation; designing and specifying the overall system structure emerges as a new kind of problem. Structural issues include gross organization and global control structure; protocols for communication, synchronization, and data access; assignment of functionality to design elements; physical distribution; composition of design elements; scaling and performance; and selection among design alternatives.</td>
<td>1993</td>
</tr>
<tr>
<td>(Kazman &amp; al, 1994)</td>
<td>The architectural design of a system can be described from (at least) three perspectives -- functional partitioning of its domain of interest, its structure, and the allocation of domain function to that structure.</td>
<td>1994</td>
</tr>
<tr>
<td>(Garlan &amp; Perry, 1995)</td>
<td>The structure of the components of a program/system, their interrelationships, and principles and guidelines governing their design and evolution over time.</td>
<td>1995</td>
</tr>
</tbody>
</table>
| (Boehm & al, 1995)         | A software system architecture comprises  
• A collection of software and system components, connections, and constraints.  
• A collection of system stakeholders' need statements.  
• A rationale which demonstrates that the components, connections, and constraints define a system that, if implemented, would satisfy the collection of system stakeholders' need statements. | 1995  |
| (Hofmeister & al, 1995)    | Software architecture has at least four distinct incarnations: Within each category, the structures describe the system from a different perspective:  
• The conceptual architecture describes the system in terms of its major design elements and the relationships among them.  
• The module interconnection architecture encompasses two orthogonal structures: functional decomposition and layers.  
• The execution architecture describes the dynamic structure of a system.  
• The code architecture describes how the source code, binaries, and libraries are organized in the development environment. | 1995  |
| (IEEE Std., 1990)          | Architecture is the organizational structure of a system.                                                                                                                                                  | 1990  |
| (Perry & Wolf, 1992)       | Software architecture is a set of architectural (or, if you will, design) elements that have a particular form.                                                                                              | 1992  |
| (Garlan & Perry, 1995)     | Software architecture is "the structure of the components of a program/system, their interrelationships, and principles and guidelines governing their design and evolution over time."                                 | 1995  |

3. Quality Attributes:

Software quality is defined as the degree to which software possesses a desired combination of attributes (Bengtsson et al., 2004). The architecture is created based on a set of requirements that it has been to full files. A quality attribute can be defined as a property of a software system (Bass et al., 2003). A quality requirement is a requirement that is placed on a software system by a stakeholder; a quality attribute is what the system actually presents once it has been implemented. In this section we will describe some of system quality attributes (Jiao et al., 2008; Boer and Vliet, 2009).

Reliability:

The capability of the software product to maintain a specified level of performance when used under specified conditions (Bass et al., 2003; Immonen and Niemela, 2009).

Usability:

Concerned with how easy it is for the user to accomplish a desired task and the kind of user support the system provides (Bass et al., 2003).

Maintainability:

Maintainability is the capability of the software product to be modified. Modifications may include corrections, improvements or adaptations of the software to changes in environment, and in requirements and functional specification (Bengtsson et al., 2004).

Testability:

The capability of the software product to enable modified software to be validated (Bass et al., 2003).

Portability:

The capability of the software product to be transferred from one environment to another. The environment
may include organizational, hardware or software environment, meaning that the system is not bounded to a specific platform/location to offer its service (Pijpers and Gordijn, 2008).

**Reusability:**
Stating that an element can be easily reused, it’s necessary that elements agree on the type of received and produced objects. Reusability depends on degree of dependency among components and it is better to be less (Pijpers and Gordijn, 2008).

**Modifiability:**
The modifiability of a software system is the ease with which it can be modified to changes in the environment, requirements or functional specification.

**Performance:**
Performance is defined as the degree to which a system or component accomplishes its designated functions within given constraints, such as speed, accuracy, or memory usage (Mattsson et al., 2006).

**Security:**
Is a measure of the system’s ability to resist unauthorized usage while still providing its services to legitimate users (Bass et al., 2003).

**Availability:**
Availability is measured as the limit of the probability that the system is functioning correctly at time \( t \), as \( t \) approaches infinity. This is the Steady-State availability of the system. It may be calculated as:

\[
\alpha = \frac{MTTF}{MTTF + MTTR}
\]

Where MTTF is the mean time to failure, and MTTR is the mean time to repair.

During the development of the architecture it is therefore important to validate that the architecture has the required quality attributes, this is usually done using one or more architecture evaluations. Quality attributes of large systems can be highly limited by a system’s requirements and constraints. Thus, it is in our best interest to try to determine as early as possible whether the system will have the desired qualities. Quality requirements should be described concretely before an architecture is developed (Barbacci et al., 2003; Immonen and Niemela, 2009).

4. **Software Architecture Styles:**

The software architecture conforms to some style. Therefore, since every software system has an architecture, every software system has a style; and styles must have existed since the first software system was developed (Giesecke et al., 2006; Garlan et al., 2009).

Clements presented below definition for architecture style (Clements et al., 2003): “An architecture style is an assignment of kind of elements and relationship among them, along a set of rules and constraints about the way of using them”. 

An architectural style includes a description of component types and their topology, a description of the pattern of data and control interaction among the components and an informal description of the benefits and drawbacks of using that style.

Architectural styles are important since they differentiate classes of designs by offering experiential evidence of how each class has been used along with qualitative reasoning to explain why each class has certain properties.

In this section we will describe some of architecture styles.

**Pipe & Filter Style:**
In the case of an application which follows the pipe and filter architecture style, each component has a set of inputs and outputs. The main characteristics of this style include the condition that filters must be independent entities (Gokhale and Yacoub, 2006).

In a pipe-and-filter system the data flow in the system is in focus. There are a number of computational components, where output from one component forms the input to the next. In its purest form, the different components are completely separated (they share no data or state), and may start processing as soon as input starts arriving. This style fits a program analyzing and formatting text or data, but is not so useful for an interactive system. Because data is copied (at least in the pure pipe-and-filter form) from outputs to inputs, performance is generally decreased (Land, 2002).
Object-Oriented Architecture:

With an object-oriented architecture, the focus is on the different items in the system, modeled as objects, classes etc. Object orientation is one of the most widely spread architectural styles, both in education, industrial practice and science. It can be discussed whether object-orientation is an architectural style or belongs to lower levels of design (Land, 2002).

Blackboard Style:

A blackboard (or repository) architecture draws the attention to the data in the system. There is a central data store, the blackboard, and agents writing and reading data. The agents may be implicitly invoked when data changes, or explicitly by some sort of external action such as a user command. A database can easily be described by the blackboard architectural style, where the blackboard itself of course is the data in the database (Land, 2002).

Layered Style:

A layer architecture, focus is laid on the different abstraction levels in a system, such as the software in a personal computer. A stack of boxes or a number of concentric circles is often used to represent a layered architecture graphically (Land, 2002).

Each successive layer is built on its predecessor, hiding the lower layer and providing some services that the upper layers make use of.

In its pure form, communications between the different layers must only occur in the interfaces between two adjacent layers. The style’s major drawbacks are that it is not always easy to identify the appropriate abstraction levels. It might also be the case that the system must communicate in a more complex way than is implicated by the layering, due to performance considerations (Land, 2002).

Table 2: Comparison of software architecture styles characteristics.

<table>
<thead>
<tr>
<th>Style</th>
<th>Sub style</th>
<th>Goal</th>
<th>Component</th>
<th>Connector</th>
<th>Advantage</th>
<th>Disadvantage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data-Centered</td>
<td>Repository/Blackboard</td>
<td>Scalability Modifiability Integrity</td>
<td>A memory/many pure computational processes</td>
<td>Computational units, interacting with memory</td>
<td>Reusable components/knowledge resources Supporting changes/maintainability</td>
<td>Low testability/lack of deterministic algorithm</td>
</tr>
<tr>
<td>Blackboard</td>
<td>Repository</td>
<td>Reusability Scalability</td>
<td>Computational Units Code Units</td>
<td>Called Procedures with state</td>
<td>Sharing large amount of data/data sharing model is publishing data</td>
<td>Subsystems must agree on specific repository/Data evolution is rigid and expensive/</td>
</tr>
<tr>
<td>Data Flow</td>
<td>Batch/Sequential/ Pipe &amp;Filter</td>
<td>Reusability Modifiability</td>
<td>Modules</td>
<td>Kinds of Connections</td>
<td>Reusability/Modifiability</td>
<td>Low performance/lack of capability in sending needed information</td>
</tr>
<tr>
<td>Pipes &amp; Filter</td>
<td>Sequential/Parallel</td>
<td>Reusability Modifiability</td>
<td>Filters</td>
<td>Pipes</td>
<td>Lack of master control/Reusability/Flexibility/Concurrency/Parallelism</td>
<td>Interactive systems designing is impossible/propagation of data in pipes</td>
</tr>
<tr>
<td>Virtual Machine</td>
<td>Interpreter/Rule-Based Systems</td>
<td>Portability</td>
<td>Embedded Procedures</td>
<td>Procedures Call</td>
<td>High portability/simulation/Adaptability</td>
<td>Low performance</td>
</tr>
<tr>
<td>Main Program/Sub Routine</td>
<td>Hierarchy</td>
<td>Modifiability</td>
<td>Procedures and visible data</td>
<td>Procedures Call/Visible data sharing</td>
<td>Shared resource with sequential access/Natural data flow/Independent process algorithms</td>
<td>Data format changing affect on all modules/Major changes is very difficult/Adding new modules affect on all modules</td>
</tr>
<tr>
<td>Object-Oriented</td>
<td>Data Oriented</td>
<td>Performance/ Reusability Modifiability Scalability</td>
<td>Objects</td>
<td>Messages</td>
<td>Implementation can be changed/Reusability</td>
<td>Adding new tasks is very hard/identification is hard</td>
</tr>
<tr>
<td>Layered</td>
<td>n-Tier Layer</td>
<td>Portability Reusability Modifiability</td>
<td>Services</td>
<td>Communication Protocols</td>
<td>Sequence order with respect to different levels/Support next generation/Dependency between services</td>
<td>Completion of all layer is rigid/abstract level definition is hard/results have low performance</td>
</tr>
<tr>
<td>Event Systems</td>
<td>Implicit Invocation/Explicit Invocation</td>
<td>Reusability Modifiability</td>
<td>Event subscriber processes</td>
<td>Automated calls/Processes</td>
<td>Parallel Procedure call/System evolution &amp; development</td>
<td>Systems control is complex</td>
</tr>
</tbody>
</table>

Table 2: Comparison of software architecture styles characteristics.
Table 2 presents comparison of software architecture styles characteristics. Table 3 is a summary of advantages and disadvantages of using one style rather than another one in consideration to qualities expected to be the most troublesome.

**Table 3: Quality attributes related to each architecture style.**

<table>
<thead>
<tr>
<th>Architecture Style</th>
<th>Quality Attribute</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Testability</td>
<td>Scalability</td>
</tr>
<tr>
<td>Layered</td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td>Client/Server</td>
<td>-</td>
<td>+/-</td>
</tr>
<tr>
<td>Pipes &amp; Filters</td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td>Repository</td>
<td>-</td>
<td>+</td>
</tr>
<tr>
<td>Object-Oriented</td>
<td>-</td>
<td>+</td>
</tr>
<tr>
<td>Event Based</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Blackboard</td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td>Batch Sequential</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Virtual Machine</td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td>Main Program/Subroutine (RPC)</td>
<td>-</td>
<td>+</td>
</tr>
<tr>
<td>Implicit Invocation</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

If the quality is addressed in a positive way the score is “+”, if using this style will penalize the system on matter of that particular quality the score is “-“, and finally if the style is neutral towards that quality the score is “+/-”.

It is important to consider the most positive as well as the most negative influences of imposing an architectural style.

If a system has to be designed for high reusability, then the communication between different components of architecture should be loosely coupled. This causes reduction of interrelationship between components. For example, layered, pipe & filter and data-centered styles, could be good selections in this case.

If a system has to be designed for high testability, then the components should have modular features. Since each component can be tested independently, modification of a component or sub-component will not cause other components to be modified. For example, pipe & filter and object oriented styles are good selections in this case.

5. Conclusion:

Measurements of software quality attributes, is one of the important concepts in software architecture evaluation and variety of techniques are used for analyzing specific quality attributes in a system.

Promoting one quality attribute requirement usually has an adverse effect on some other quality attribute requirement.

Architectural decisions will promote some quality attribute requirements while inhibiting others, thus resulting in quality attribute tradeoff decisions. These tradeoffs are best dealt with in the earliest phases of system development-during the design of the architecture (Barbacci *et al.*, 2003).
In general, it is not possible to select an architecture style which addresses all of quality attribute requirements. In fact a specific style is suitable for some special goals and not for all purposes we may not to provide all quality attributes simultaneously.

Therefore, the selection of architecture style must be so that, there is a good trade-off between required quality attributes in system.

REFERENCES