What is your definition of software architecture?

WHAT IS YOUR DEFINITION OF SOFTWARE ARCHITECTURE?

The SEI has compiled a list of modern, classic, and bibliographic definitions of software architecture.

Modern definitions are definitions from Software Architecture in Practice and from ANSI/IEEE Std 1471-2000, Recommended Practice for Architectural Description of Software-Intensive Systems.

Classic definitions appear in some of the more prominent or influential books and papers on architecture.

Bibliographic definitions are taken from papers and articles in our software architecture bibliography.

Modern Software Architecture Definitions

Entries

1. From the book *Documenting Software Architectures: Views and Beyond (2nd Edition),* **Clements et al, Addison-Wesley, 2010:**
The set of structures needed to reason about the system, which comprises software elements, relations among them, and properties of both.

In this book, we use a definition based on the one from Software Architecture in Practice (2nd Edition) (see below). We chose it because it helps us know what to document about an architecture. The definition emphasizes the plurality of structures present in every software system. These structures, carefully chosen and designed by the architect, are the key to achieving and reasoning about the system's design goals. And those structures are the key to understanding the architecture. Therefore, they are the focus of our approach to documenting a software architecture.

Structures consist of elements, relations among the elements, and the important properties of both. So documenting a structure entails documenting those things.

2. From the book *Software Architecture in Practice* **(2nd edition), Bass, Clements, Kazman; Addison-Wesley 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.

"Externally visible" properties refers to those assumptions other elements can make of an element, such as its provided services, performance characteristics, fault handling, shared resource usage, and so on. Let us look at some of the implications of this definition in more detail.

First, *architecture defines elements.* The architecture embodies information about how the elements relate to each other. This means that architecture specifically

©2017 Carnegie Mellon University | 3854 | 01.22.17

omits certain information about elements that does not pertain to their interaction. Thus, an architecture is foremost an abstraction of a system that suppresses details of elements that do not affect how they use, are used by, relate to, or interact with other elements. In nearly all modern systems, elements interact with each other by means of interfaces that partition details about an element into public and private parts. Architecture is concerned with the public side of this division; private details of elements—details having to do solely with internal implementation—are not architectural.

Second, the definition makes clear that *systems can and do comprise more than one structure* and that no one structure holds the irrefutable claim to being the architecture. For example, all non-trivial projects are partitioned into implementation units; these units are given specific responsibilities, and are the basis of work assignments for programming teams. This kind of element will comprise programs and data that software in other implementation units can call or access, and programs and data that are private. In large projects, the elements will almost certainly be subdivided for assignment to subteams. This is one kind of structure often used to describe a system. It is a very static structure, in that it focuses on the way the system's functionality is divided up and assigned to implementation teams.

Other structures are much more focused on the way the elements interact with each other at runtime to carry out the system's function. Suppose the system is to be built as a set of parallel processes. The set of processes that will exist at runtime, the programs in the various implementation units described previously that are strung together sequentially to form each process, and the synchronization relations among the processes form another kind of structure often used to describe a system.

Third, the definition implies that *every software system has an architecture* because every system can be shown to be composed of elements and relations among them. In the most trivial case, a system is itself a single element—an uninteresting and probably non-useful architecture, but an architecture nevertheless. Even though every system has an architecture, it does not necessarily follow that the architecture is known to anyone. Unfortunately, an architecture can exist independently of its description or specification, which raises the importance of architecture documentation and architecture reconstruction.

Fourth, *the behavior of each element is part of the architecture* insofar as that behavior can be observed or discerned from the point of view of another element. This behavior is what allows elements to interact with each other, which is clearly part of the architecture. This does not mean that the exact behavior and performance of every element must be documented in all circumstances; but to the extent that an element's behavior influences how another element must be written to interact with it or influences the acceptability of the system as a whole, this behavior is part of the software architecture.

Finally, the definition is indifferent as to whether the *architecture for a system is a good one or a bad one*, meaning that the architecture will allow or prevent the system from meeting its behavioral, performance, and life-cycle requirements. Assuming that we do not accept trial and error as the best way to choose an architecture for a system—that is, picking an architecture at random, building the system from it, and hoping for the best—this raises the importance of *architecture evaluation.*

3. ANSI/IEEE Std 1471-2000, Recommended Practice for Architectural Description of Software-Intensive Systems

Architecture is defined by the recommended practice as the *fundamental organization of a system, embodied in its components, their relationships to each other and the environment, and the principles governing its design and evolution.* This definition is intended to encompass a variety of uses of the term architecture by recognizing their underlying common elements. Principal among these is the need to understand and control those elements of system design that capture the system's utility, cost, and risk. In some cases, these elements are the physical components of the system and their relationships. In other cases, these elements are not physical, but instead, logical components. In still other cases, these elements are enduring principles or patterns that create enduring organizational structures. The definition is intended to encompass these distinct, but related uses, while encouraging more rigorous definition of what constitutes the fundamental organization of a system within particular domains.

Classic Software Architecture Definitions

Rational Unified Process, 1999

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 (Kruchten: The Rational Unified Process. Also cited in Booch, Rumbaugh, and Jacobson: *The Unified Modeling Language User Guide,* Addison-Wesley, 1999).

PERRY AND WOLF, 1992

An early one by Dewayne Perry and Alex Wolf is:
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.

GARLAN AND SHAW, 1993

In what has come to be regarded as a seminal paper on software architecture , Mary Shaw and David Garlan suggest that 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."*

BASS, ET AL., 1994

Writing about a method to evaluate architectures with respect to the quality attributes they instill in a system , Bass and his colleagues write that

> *...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.*

HAYES-ROTH, 1994

Writing for the ARPA Domain-Specific Software Architecture (DSSA) program, Hayes-Roth says that software architecture is

> *...an abstract system specification consisting primarily of functional components described in terms of their behaviors and interfaces and component-component interconnections.*

GARLAN AND PERRY, 1995

David Garlan and Dewayne Perry have adopted the following definition for their guest editorial to the April 1995 IEEE Transactions on Software Engineering devoted to software architecture:

The structure of the components of a program/system, their interrelationships, and principles and guidelines governing their design and evolution over time.

(The source of this definition was a weekly discussion group devoted to software architecture at the Software Engineering Institute.)

BOEHM, ET AL., 1995

Barry Boehm and his students at the USC Center for Software Engineering write that:

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.

SONI, NORD, AND HOFMEISTER, 1995

Soni, Nord, and Hofmeister of Siemens Corporate Research write that, based on structures found to be prevalent and influential in the development environment of industrial projects they studied, 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.

Shaw, 1995: At the First International Workshop on Architectures for Software Systems, Mary Shaw provided a much-needed clarification of the terminological chaos. Distilling the definitions and viewpoints (implicit or explicit) of the workshop's position papers, Shaw classifies the views of software architecture thus :

- **Structural models** all hold that software architecture is composed of components, connections among those components, plus (usually) some other aspect or aspects, including (grouping suggested by the authors):
- configuration, style
- constraints, semantics
- analyses, properties
- rationale, requirements, stakeholders' needs

Work in this area is exemplified by the development of architectural description languages (ADLs), which are formal languages that facilitate the description of an architecture's components and connections. The languages are usually graphical, and provide some form of "box and line" syntax for specifying components and hooking them together.

- **Framework models** are similar to the structural view, but their primary emphasis is on the (usually singular) coherent structure of the whole system, as opposed to concentrating on its composition. Framework models often target specific domains or problem classes. Work that exemplifies the framework view includes domain-specific software architectures, CORBA [55] or CORBA-based architecture models, and domain-specific component repositories (e.g., PRISM).
- **Dynamic models** emphasize the behavioral quality of systems. "Dynamic" may refer to changes in the overall system configuration, setting up or disabling pre-enabled communication or interaction pathways, or the dynamics involved in the progress of the computation, such as changing data values.
- **Process models** focus on construction of the architecture, and the steps or process involved in that construction. In this view, architecture is the result of following a process script. This view is exemplified by work in process programming for deriving architectures.

These views do not preclude each other, nor do they really represent a fundamental conflict about what software architecture is. Instead, they represent a spectrum in the software architecture research community about the emphasis that should be placed on architecture -- its constituent parts, the whole entity, the way it behaves once built, or the building of it. Taken together, they form a consensus view of software architecture.

Bibliographic Software Architecture Definitions

[Lane 90]: Software architecture is the study of the large-scale structure and performance of software systems. Important aspects of a system's architecture include the division of functions among system modules, the means of communication between modules, and the representation of shared information.

[Rechtin 92]: Systems architecture: The underlying structure of a system, such as a communication network, a neural network, a spacecraft, a computer, major software or an organization.

[Bhansali 92]: A generic architecture is defined as a topological organization of a set of parameterized modules, together with the inter-modular relationships. Designing a software system using a generic architecture consists of instantiating the parameters of each parameterized module by a concrete value while maintaining the inter-modular constraints.

[Garlan 92]: As the size and complexity of software systems increases, the design problem goes 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; composition of design elements; scaling and performance; and selection among design alternatives. This is the software architecture level of design.

[Perry 92]: We distinguish three different classes of architectural elements: processing elements; data elements; and connection elements. The processing elements are those components that supply the transformation on the data elements; the data elements are those that contain the information that is used and transformed; the connecting elements (which at times may be either processing or data elements, or both) are the glue that holds the different pieces of the architecture together. For example, procedure calls, shared data, and messages are different examples of connecting elements that serve to "glue" architectural elements together. Consider the example of water polo as a metaphor for the different classes of elements: the swimmers are the

processing elements, the ball is the data element, and the water is the primary connecting element (the "glue"). Consider further the similarities of water polo, polo, and soccer. They all have a similar "architecture" but differ in the "glue"- that is, they have similar elements, shapes and forms, but differ mainly in the context in which they are played and in the way that the elements are connected together. We shall see below that these connecting elements play a fundamental part in distinguishing one architecture from another and may have an important effect on the characteristics of a particular architecture or architectural style. The architectural form consists of weighted properties and relationships. The weighting indicates one of two things: either the importance of the property or the relationship, or the necessity of selecting among alternatives, some of which may be preferred over other. The use of weighting to indicate importance enables the architect to distinguish between "load-bearing" and "decorative" formal aspects; the use of weighting to indicate alternatives enables the architect to constrain the choice while giving a degree of latitude to the designers who must satisfy and implement the architecture. Properties are used to constrain the choice of architectural elements-that is, the properties are used to define constraints on the elements to the degree desired by the architect. Properties define the minimum desired constraints unless otherwise stated- that is, the default on constraints defined by properties is: "what is not constrained by the architect may take any form desired by the designer or implementer. "Relationships are used to constrain the "placement" of architectural elements-that is, they constrain how the different elements may interact and how they are organized with respect to each other in the architecture. As with properties, relationships define the minimum desired constraints unless otherwise stated. An underlying, but integral, part of an architecture is the rationale for the various choices made in defining an architecture. The rationale captures the motivation for the choice of architectural style, the choice of elements, and the form.

[Crispen 94]: An Architecture, as we intend to use the term, consists of (a) a partitioning strategy and (b) a coordination strategy. The partitioning strategy leads to dividing the entire system in discrete, non-overlapping parts or components. The coordination strategy leads to explicitly defined interfaces between those parts.

[Clements 94-2]: Software architecture is loosely defined as the organizational structure of a software system including components, connections, constraints, and rationale. Components can be small pieces of code, such as modules, or larger chunks, such a stand-alone programs like database management systems. Connections in an architecture are abstractions for how components interact in a system, e.g., procedure calls, pipes, and remote procedure calls. An architecture has various constraints and rationales associated with it, including the constraints on component selection and the rationale for choosing a specific component in a given situation.

[Moriconi 94]: A software architecture is represented using the following concepts: 1. Component: An object with independent existence, e.g., a module, process, procedure, or variable. 2. Interface: A typed object that is a logical point of interaction between a component and its environment. 3. Connector: A typed object relating interface points, components, or both. 4. Configuration: A collection of constraints that wire objects into a specific architecture. 5. Mapping: A relation between the vocabularies and the formulas of an abstract and a concrete architecture. The formula mapping is required because the two architectures can be written in different styles. 6. Architectural style: A style consists of a vocabulary of design elements, a set of well-formedness constraints that mush be satisfied by any architecture written in the style, and a semantic interpretation of the connectors. Components, interfaces, and connectors are treated as first-class objects- i.e., they have a name and they are refinable. Abstract architectural objects can be decomposed, aggregated, or eliminated in a concrete architecture. The semantics of components is not considered part of an architecture, but the semantics of connectors is.

[Kruchten 94]: Software architecture deals with the design and implementation of the high-level structure of the software. It is the result of assembling a certain number of architectural elements in some well-chosen forms to satisfy the major functionality and performance requirements such as scalability and availability. Software architecture deals with abstraction, with decomposition and composition, with style and aesthetics.

[Garlan 94]: A critical aspect of the design for any large software system is its gross structure that is, its high-level organization of computational elements and interactions between those elements. Broadly speaking, we refer to this as the software architectural level of design.

[FHayes-Roth 94]: Software Architecture: An abstract system specification consisting primarily of functional components described in terms of their behaviors and

interfaces and component-component interconnections. The interconnections define provide by which components interact.

[Abowd 95]: Software architecture is an important level of description for software systems. At this level of abstraction key design issues include gross-level decompositional components, protocols of interaction between those components, global system properties (such as throughout and latency), and life-cycle issues (such as maintainability, extent of reuse, and platform independence).

[Boasson 95]: [We take] "architecture" to mean a system structure that consists of active modules, a mechanism to allow interaction among these modules, and a set of rules that govern the interaction.

[Garlan 95]: The structure of the components of a program/system, their interrelationships, and principles and guidelines governing their design and evolution over time.

[BHayes-Roth 95]: The architecture of a complex software system is its "style and method of design and construction".

[Lawson 95]: A system architecture is typically defined in the context of the "requirements, design, implementation" sequence, referring the top level of the design stage, "...where the design issues involve overall association of systems capabilities with components." It also designates a higher level of abstraction, codification, and standardization, targeting the improvement of system design and making the complex system intellectually tractable. To characterize a system architecture the following topics must be addressed: The relations that bind a system architecture to the corresponding development process: the important decisions to be made (at the corresponding level of abstraction), the issues to be resolved, the properties to be guaranteed. The relations to the information model employed, and the tools used. The corresponding body of applicable engineering knowledge and design rationale. The set of constructive concepts-architecture elements- as well as notations for them, that can be used to build the system description at the appropriate level of abstraction. From this perspective, we define and architecture as a system design model that captures system organization and behavior in terms of components, interactions, and static and dynamic configurations.

[KJackson 95]: "The definition of a set of generic component types together with: -a description of the properties of each type, -the rules governing the way each component type may interact with each other type -the style of interactions allowed between components, and -the rules which govern how a system (or subsystem) may be composed from instances of the generic components." For an architecture to be considered "good" and provide the facilities defined in the previous section an architecture must, in addition to the items indicated above: -support the specification of design specific but context independent (reusable) component types; -support the composition of system (and subsystems) from instances of these design specific components, -support the ability to place components in appropriate physical locations and define the run-time software and/or special purpose hardware required to support the execution of the system. (We refer to this item as "infrastructure" or "middleware", since it sits between the application software and the standard computer hardware plus operating system.

[ATA 96]: The U.S. Army's Army Technical Architecture (ATA) provides these definitions:

- A Technical Architecture is the minimal set of rules governing the arrangement, interaction, and interdependence of the parts or elements that together may be used to form an information system. Its purpose is to ensure that a conformant system satisfies a specified set of requirements. It is the build code for the Systems Architecture being constructed to satisfy Operational Architecture requirements.
- An Operational Architecture is a description, often graphical, which defines the force elements and the requirement to exchange information between those force elements. It defines the types of information, the frequency of its exchange, and what warfighting tasks are supported by these information exchanges. It specifies what the information systems are operationally required to do and where these operations are to be performed.
- A Systems Architecture is a description, often graphical, of the systems solution used to satisfy the warfighter's Operational Architecture requirement. It defines the physical connection, location, and identification of nodes, radios, terminals, etc., associated with information exchange. It also specifies the system performance parameters. The Systems Architecture is constructed to satisfy the Operational Architecture requirements per the standards defined in the Technical Architecture