Software Architecture in Practice

Chapter 2: What Is Software Architecture? Why Is It Important?

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213-3890

Sponsored by the U.S. Department of Defense

Lecture Objectives

This lecture will

- **introduce and define the term "software architecture"**
- **explain the value that a software architecture brings to a development project**
- **describe how software architecture is composed of several different software structures**
- **give examples of several commonly used architectural structures and show their uses**

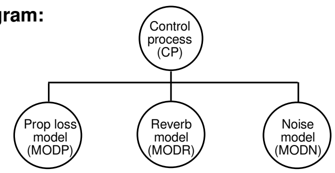
What Is Software Architecture?

Some Usual Descriptions of Architecture

"Components and connectors"

"Overall structure of system"

A diagram:



What's Wrong with "Components and Connectors?"

What kind of component?
- **task? process?**
- **object? program? function?**
- **library? compilation unit?**
- **processor?**

What kind of connector?
- **calls? invokes? signals? uses? data flow?**
- **subclass?**
- **runs with? excludes?**
- **co-located with?**

What's Wrong with "Overall Structure?"

***Which* structure? Software is composed of *many* structures.**
- **module**
- **task**
- **uses**
- **logical**
- **functional**

When seeing boxes and lines, we must ask
- **What do the boxes represent?**
- **What do the arrows mean?**

What's Wrong with the Diagram?

Same questions as the previous slide.
- **What kind of components?**
- **What kind of connectors?**
- **What structures?**
- **What do the boxes and arrows mean?**

Plus new questions
- **What is the significance of the layout?**
- **Why is control process on a higher level?**

Box and arrow drawings alone are not architectures; rather, they are a starting point.

The Definition of Software Architecture

The software architecture of a program or computing system is the structure or structures of the system, which comprise software components, the externally visible properties of those components, and the relationships among them.

Implications of Our Definition - 1

Architecture is an abstraction of systems.
- **Architecture defines components and how they interact.**
- **Architecture suppresses purely local information about components; private details are not architectural.**

Systems have many structures (views).
- **No single structure can be *the* architecture.**
- **The set of candidate structures is not fixed or prescribed: whatever is useful for analysis, communication, or understanding.**

Implications of Our Definition -2

Properties of components are assumptions that one component can make of another. Properties include

- provided services (functionality)
- required services
- performance characteristics
- fault handling
- shared resource usage

Implications of Our Definition -3

This means that box-and-line drawings alone are *not* architectures, but a starting point.

- You might *imagine* the behavior of a box labeled “database” or “executive.”
- You need to add specifications and properties.

Implications of Our Definition -4

Relationships are more general than *connectors*.

Relationships may be runtime (connectors).

- sends data to
- invokes
- signals

Relationships may also be non-runtime.

- is a submodule of
- encapsulates information
- inherits from

Implications of Our Definition -5

Every system *has* an architecture.
- **Every system is composed of components and relationships among them.**
- **In the simplest case, a system is composed of a single component, related only to itself.**

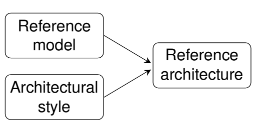
Just having an architecture is different from having an architecture that is known to everyone. New issues:
- **architecture versus specification of the architecture**
- **architecture recovery and conformance**
- **rationale for the architecture**

Related Concepts

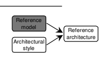
Reference model

Architectural style

Reference architecture



Reference Model

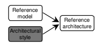


Reference model: a division of functionality together with data flow between the pieces

A reference model divides a problem into pieces, but does not prescribe a software solution.

Example: If you can name the standard parts of a compiler, an operating system, or a database management system, it is because you have been taught a reference model.

Architectural Style - 1

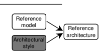


Architectural style: ***a description of component and connector types and a pattern of their runtime control and/or data transfer.*** **[Shaw 96]**

Architectural styles are a set of canonical architectural solutions to problems.

Styles are underspecified architectures. They suggest patterns of runtime interaction and topologies of components.

Architectural Style - 2

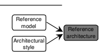


A style may be thought of as
- **a set of constraints on an architecture**
- **an abstraction for a set of related architectures**

Styles appearing in the literature include
- **client-server**
- **cooperating process**
- **data-centered**
- **layered**

We will discuss styles later.

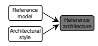
Reference Architecture - 1



Reference architecture: ***a reference model mapped onto software components (that will cooperatively implement the functionality defined in the reference model) and the data flows between the components***

A reference architecture allows
- **definition of software development teams**
- **searching for preexisting components**
- **allocation of expertise and other resources**
- **estimates of cost and schedule**

Reference Architecture -2

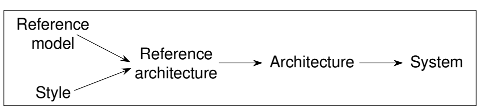


Alternative meaning: the (common) architecture for a related group of systems

But, every architecture applies across a group of systems, so this definition is not very helpful.

Reference Architecture -3

Building a system is a process. At each successive phase in the process, more requirements are addressed, and more design and development have taken place.



Why Is Architecture Important? (Technical Aspects)

Architecture is important for three primary reasons.

1. **It provides a vehicle for communication among stakeholders.**
2. **It is the manifestation of the earliest design decisions about a system.**
3. **It is a transferable, reusable abstraction of a system.**

Communication Vehicle

Architecture provides a common frame of reference in which competing interests may be exposed and negotiated.

- negotiating requirements with users
- keeping the customer informed of progress and cost
- implementing management decisions and allocations

Result of Early Design Decisions - 1

An architecture defines constraints on an implementation.

- implementations must conform to architecture
- (global) resource allocation decisions constrain implementations of individual components
- system trade-offs are in the architectural realm

Result of Early Design Decisions - 2

The architecture dictates organizational structure for development/maintenance efforts. Examples include

- division into teams
- units for budgeting, planning
- basis of work breakdown structure
- organization for documentation
- organization for CM libraries
- basis of integration
- basis of test plans, testing
- basis of maintenance

Once decided, architecture is extremely hard to change!

Result of Early Design Decisions -3

Architecture permits/precludes achievement of a system's desired quality attributes. For example:

If you desire	Examine
performance	inter-component communication
modifiability	component responsibilities
security	inter-component communication, specialized components (e. g., kernels)
scalability	localization of resources
ability to subset	inter-component usage
reusability	inter-component coupling

The architecture influences qualities, but does not guarantee them.

Result of Early Design Decisions -4

Since architecture influences qualities in known ways, it follows that we can use architecture to predict how well quality attributes will be achieved.

We can analyze an architecture for achievement of quality attributes.

We will discuss this in more detail later.

Result of Early Design Decisions -5

An architecture helps users reason about and manage change (about 80% of effort in systems occurs *after* deployment).

Architecture divides all changes into three classes.
- **local: modifying a single component**
- **non-local: modifying several components**
- **architectural: modifying the gross system topology, communication, and coordination mechanisms**

A good architecture is one in which the most likely changes are also the easiest to make.

Result of Early Design Decisions -6

An architecture helps with evolutionary prototyping.

- Architecture serves as a skeletal framework into which components can be plugged.
- By segregating functionality into appropriate components, experimentation is easier.

Result of Early Design Decisions -7

To summarize, architecture establishes the path for development. Architecture

- constrains the implementation (developers are constrained by architecture)
- organizes the development effort
- provides a first approach at achieving quality requirements
- helps manage change
- helps with prototyping

The right architecture makes things go smoothly. The wrong architecture leads to disaster.

Reusable Model -1

An architecture is an abstraction: a one-to-many mapping (one architecture, many systems).

Architecture is the basis for product (system) commonality. Entire product lines can share a single architecture.

Systems can be built from large, externally developed components that are tied together via architecture.

Reusable Model -2

A component's functionality can be separated from its interconnection mechanisms.

Often, a component's packaging makes it difficult to reuse a component. For example, if you need
- an object, you can't use a task
- a task, you can't use an object
- to invoke with a pipe, you can't use a called program
- a program, you can't use a file

Reusable Model -3

Less is more: It pays to restrict the vocabulary of design alternatives.

Architectural styles serve as that restricted vocabulary of design alternatives.

Working with known styles
- reduces learning time
- enhances communication
- takes advantage of known style properties (e.g., performance, security, reliability)

Reusable Model -4

Architectures can enable template-based development.

Templates may be used to code component interaction frameworks. The developer fills in the unique part, and reuses the common part.

Templates enhance
- speed of development
- reliability
  - source of many errors eliminated
  - fixing one error improves many places

Reusable Model -5

In summary, an architecture forms a reusable model.
- **enables product lines**
- **enables systems to be built from externally developed components**
- **separates functionality from interconnection mechanisms**
- **provides a vocabulary of design**
- **enables template-based component development**

Architectural Structures -1

In a house, there are plans for
- **rooms**
- **electrical wiring**
- **plumbing**
- **ventilation**

Each of these constitutes a "view" of the house.
- **used by different people**
- **used to achieve different qualities in the house**
- **serves as a description and prescription**

So it is with software architecture.

Architectural Structures -2

Which structures are used, and why?

Common structures include
- **module**
- **process**
- **uses**
- **calls**
- **data flow**
- **class**
- **physical**

A structure provides a *view* of the architecture.

Module Structure

Components: modules, work assignments

Relations: "is a submodule of," "shares a secret with"

Used: as a basis of team structure and resource allocation

Affected attributes include: maintainability, understandability

Process Structure

Components: tasks, processes

Relations: "synchronizes with," "excludes," "preempts"

Used: to tune system runtime performance, exploit multiprocessing hardware

Affected attributes include: performance

Uses Structure

Components: procedures

Relations: "assumes the correct presence of"

Used: to engineer subsets, supersets

Affected attributes include: reusability, testability, incremental development

Calls Structure

Components: procedures

Relation: invokes

Used: to trace control flow; for debugging

Affected attributes include: buildability, testability, maintainability, understandability

Data Flow Structure

Components: programs, modules

Relation: "may send data to"

Used: for traceability of functionality

Affected attributes include: performance, correctness, accuracy

Class Structure

Components: objects

Relation: "inherits from," "is instance of"

Used: to exploit similarity among objects

Affected attributes include: development time, maintainability

Physical Structure

Components: tasks, processes, processors

Relation: "resides on same processor"

Used: to manage process-to-processor allocation

Affected attributes include: performance, availability

What Are Structures Used For?

Documentation vehicle for
- current development
- future development
- managers
- customers

Engineering tool to help achieve qualities

Architectural Structures Summary

Structures are related to each other in complicated ways.

In some systems, different structures collapse into a single one. (For example, process structure may be the same as module structure for extremely small systems.)

Lesson: Choose the structures that are useful to the system being built and to the achievement of qualities that are important to you.

Lecture Summary

Architecture is important because
- it provides a communication vehicle among stakeholders
- it is the result of the earliest design decisions
- about a system

An architecture is composed of many structures, which are software components and their relationships.

Each structure provides engineering leverage on different qualities. Engineer and document the structures that help to achieve your desired qualities.

Discussion Questions -1

1. Software architecture is often compared to the architecture of buildings as a conceptual analogy. What are the strong points of that analogy? What is the correspondence in buildings to software architecture structures and views? To styles? What are the weaknesses of the analogy? When does it break down?

2. What's the difference between a reference architecture and an architectural style? What can you do with one that you can't do with the other in terms of (a) organizational planning and (b) architectural analysis?

Discussion Questions -2

3. Do the architectures in your organization recognize the different views (structures and relations) inherent in architectures? If so, which ones? If not, why not?

4. Is there a different definition of software architecture that you are familiar with? If so, think about the ways in which this definition supports our acid test of an architecture: Does it abstract away information from the system and yet provide enough information to be a basis for analysis, decision making, and risk reduction?