Software Architecture for Dependable Systems

Part 1: Concepts and Principles of Software Architecture

David Garlan
Carnegie Mellon University

Examples of Software Architecture

More Examples

More Examples

More Examples

More Examples

© David Garlan, 2008
Descriptions of Software Architecture

- Descriptions often include a section on "the architecture of this system"
- Usually informal prose plus box-and-line diagram
- Lots of appeal to intuition
- Little precision, rarely formal

The Challenge

- Turn Software Architecture into an engineering discipline
- From ad hoc definition to codified principles
- Develop systems "architecturally"
  - Build systems compositionally from parts
  - Assure that the system conforms to the architecture and has the desired properties
  - Use standard integration architectures
  - Reuse codified architectural design expertise
  - Reduce costs through product-lines

My Lectures

- Introduction to Software Architecture: Concepts and Principles
- Modeling and Analysis of Software Architectures
- Practical Topics: Tools, Conformance, Self-Healing
The Big Problem

Requirements

???

Implementations

How to bridge the gap between requirements and solutions?

One Possible Answer

Requirements

A Miracle Happens!

Implementations

- Ad hoc
- Requires gurus
- Unpredictable
- Costly

The Role of Software Architecture

Software Architecture

Requirements

- High level of system design
- System-level abstractions
- Reuse design idioms

Implementations

What is Software Architecture?

☐ There are many definitions in the literature
  - CMU’s Software Engineering Institute’s web site on software architecture lists over 70
  - Most definitions look like this:

A software architecture for a system is the structure or structures of the system, which comprise elements, their externally-visible properties, and the relationships among them.

Issues Addressed by Software Architecture

☐ Gross decomposition of a system into parts
  - often using rich abstractions for component interaction (or system “glue”)
  - often using common design patterns/styles

☐ Emergent system properties
  - performance, throughput, latencies
  - reliability, security, fault tolerance, evolvability

☐ Rationale
  - justifying architectural decisions

☐ Envelope of allowed change
  - “load-bearing walls”

Google File System

Figure 1: GFS Architecture

Source: “The Google File System”

Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung.

SOSP 2003.
2008 Lugano Summer School
SW Arch. for Dependable Systems

David Garlan
Carnegie Mellon University

Architectural Design Task
Different issues for architecture & programs

<table>
<thead>
<tr>
<th>Architecture</th>
<th>Programs</th>
</tr>
</thead>
<tbody>
<tr>
<td>interactions among parts</td>
<td>implementations of parts</td>
</tr>
<tr>
<td>structural properties</td>
<td>computational properties</td>
</tr>
<tr>
<td>declarative</td>
<td>operational</td>
</tr>
<tr>
<td>mostly static</td>
<td>mostly dynamic</td>
</tr>
<tr>
<td>system-level performance</td>
<td>algorithmic performance</td>
</tr>
<tr>
<td>outside module boundary</td>
<td>inside module boundary</td>
</tr>
</tbody>
</table>

Why Should You Care?

- Reduce development and maintenance costs
  - Reuse of designs
  - Improve understandability
- Improve quality of product
  - Clarify requirements
  - Make principled engineering decisions
  - Early analysis of design flaws

But How Much Architecture is Enough?


Key Points
- Projects have a "sweet spot" of investment into "front-end" activities, including architectural design
  - Too little => more rework later
  - Too much => wasted effort
- Amount of depends on size of system
  - Small (10 KLoC) => 5-10%
  - Medium (100 KLoC) => 20-25%
  - Large (10 MLoC) => 35-40%

Evolution of the Field of Software Architecture – 1980’s

- Informal use of box and line diagrams
- Ad hoc application of architectural expertise
- Diverse, uncodified use of architectural patterns and styles
- No identified “architect” on most projects

© David Garlan, 2008
1990’s

- Recognition of the value of architects in software development organizations
- Processes requiring architectural design reviews & explicit architectural documentation
- Use of product lines, commercial architectural standards, component integration frameworks
- Codification of vocabulary, notations & tools for architectural design
- Books/courses on software architecture

2000’s

- Incorporation of architectural notions into mainstream design languages (e.g., UML-2), and tools (e.g., Rose)
- Methods based on architectural design and refinement (e.g., Model-Driven Architecture – MDA)
- Some architecture analysis tools
- Architectural standards for Enterprise Systems (e.g., RM-ODP, TOGAF)

Concepts

- Views and Viewtypes
  - Why they are necessary
  - What distinguishes them
- Styles & Patterns
  - Why they are necessary
  - What distinguishes them
- The Module Viewtype
  - Elements
  - What it’s good for
  - Layered styles
- The Component and Connector Viewtype
  - Elements
  - What it’s good for
  - Styles overview

System Structure

- Recall our working definition
  - A software architecture for a system is the structure or structures of the system, which comprise elements, their externally-visible properties, and the relationships among them.
- But what kinds of structure?
  - modules, showing composition/decomposition
  - processes, and how they synchronize
  - hardware and how software is deployed on it
  - runtime components
- Each is the basis of an Architectural View

Structures and Views – 1

A human body is comprised of multiple structures.
One body has many structures, and those structures have many views. So it is with software...

Structures And Views – 2

These views are needed by the cardiologist...
...but will these views work for the orthopedist?
Different stakeholders are interested in different structures.
Views must represent the structures that the stakeholders are interested in. So it is with software...
Views

- A view is a representation of a set of system elements and the relations associated with them.
- Not all system elements -- some of them.
- A view selects element types and relation types of interest, and shows those.

What Views Are Available? -1

- Plenty! Too many!
- An architect needs a way to choose the useful ones
- One thing that would help is to organize the views into broad categories

What Views Are Available? -2

- An architect must consider the system in three ways:
  1. How is it structured as a set of code units?
  2. How is it structured as a set of elements that have run-time behavior and interactions?
  3. How does it relate to non-software structures in its environment?

What Views Are Available? -3

This suggests looking for three kinds of views:
1. How it is structured as a set of code units module views (Module Viewtype)
2. How it is structured as a set of elements that have run-time behavior and interactions component-and-connector views (C&C Viewtype)
3. How it relate to non-software structures in its environment? allocation views (Allocation Viewtype)

What Views Are Available? -4

- Views in the Module Viewtype show elements that are units of implementation.
- Views in the Component-and-Connector Viewtype (C&C) show elements that have run-time behavior and interaction.
  (There is often, but not always, a correspondence between the two.)
- Views in the Allocation Viewtype show how software structures are allocated to non-software structures.

Example: Alternating Characters
Code (Module) View

Produce alternating case of characters in a stream

Legend

Definition/Use Modularization

© David Garlan, 2008
This still leaves us with enormous latitude. Within each viewtype, recurring forms have been widely observed in different systems. These forms, or patterns, are worth capturing because they have known properties and can be re-used. We call these styles:

An architectural style is a specialization of element and relation types, together with a set of constraints on how they can be used.

When experts work on a problem it is unusual for them to invent new solutions. They recall a similar problem and reuse the essence of the solution. Abstraction and codifying specific problem-solution pairs and distilling the common features leads to styles or patterns. Patterns can help designers and builders leverage the collective experience of a community of designers and builders.

Styles are widely known in the literature. They represent "tools" in the architect's "bag of tricks." Note that styles exist independent of any system. Two different systems can use the same style. A system is likely to be composed of more than one style. Different "areas" of the system exhibit different styles. The same system might simply be seen in different lights, as though through filtered glasses.

Viewtypes reflect the three broad ways an architect must consider a system: units of implementation (module viewtype) run-time units (C&C viewtype) relation to non-software structures (allocation viewtype) Even within a viewtype, many choices remain: how elements are restricted, how they are related to each other, how they are used or configured. These choices, when made, yield styles.

Elements: Modules. A module is a code unit that implements a set of responsibilities.

Relations: Relations among modules include A is part of B. This defines a part-whole relation among modules. A depends on B. This defines a dependency relation among modules. A is a B. This defines specialization and generalization relations among modules.
What Are Module Views Used For?

- **Construction:** These are the blueprints for the code. Modules are assigned to teams for implementation. Modules are often the unit for subsequent design (e.g., of interfaces).
- **Analysis:** Traceability and impact analysis rely on implementation units. Project management, budgeting, planning, and tracking often use modules.

Notations for Module Views

- Informal: box-and-line, with nesting
- UML: Class diagrams

![Diagram of UML class diagrams](image)

Module Styles

- **Decomposition Style**
  - Hierarchical decomposition of modules
  - Supports concurrent development
- **Generalization Style**
  - Specialization hierarchy
  - Supports reuse; managing large numbers of definitions
- **Layered Style**
  - Virtual machines
  - Supports portability, reuse

Decomposition Style

- **Elements:** Modules
- **Relations:**
  - Is part of. Criteria for decomposition vary:
    - achievement of modifiability
    - build vs. buy
    - product lines
- **What it's for:**
  - A starting point. Many architects assign functions to modules as a prelude to detailed design.
  - Change/impact analysis
  - Basis for work assignments

Generalization Style

- **Elements:** Modules
- **Relations:** Generalizes, an "is a" relation
- **Properties:** Interface or implementation inheritance?
- **What it’s for:**
  - Basis for object-oriented designs
  - Providing for evolution and extension
  - Reuse

Decomposition Style: Example

![Example of decomposition style](image)

Generalization Style

![Example of generalization style](image)
Generalization Style: Notations

- Formal:
  - Programming languages
  - UML

Layered Style

- Elements: Layers, a virtual machine
- Relation:
  - Allowed-to-use, a specialization of depends-on
- Rules:
  - Every piece of software is assigned to exactly one layer.
  - Software in a layer is allowed to use software in (any lower layer, next lower layer)
  - Software in a layer {is, is not} allowed to use other software in same layer.

Layered Style (cont’d.)

- What it’s for
  - Portability
  - Fielding subsets, incremental development
  - Separation of concerns
- Variations (many)
  - Segmented layers: Dividing a layer into segments (or sub-modules), with allowed-to-use relations between the segments within a layer and segments between layers.

Example 1: JavaPhone

- Java Applications
- Telephony HW
- Telephony HW (POTS Card/Fax Card)
- Java Telephony API
- Java Run-Time
- XTL
- TSAPI
- TAPI
- Other APIs

Example 2

- Function driver
- Shared services
- Device interfaces
- Application data types
- Extended computer
- Data bank
- Physical models
- Filter behaviors
- Software utilities

Layered Style: Informal notations

- Boxes and arrows
- Stacks of boxes
- Concentric Rings

What does this mean?

© David Garlan, 2008
Component-and-connector (C&C) Viewtype

- **Elements:**
  - Components: principal units of run-time computation and data stores
  - Connectors: interaction mechanisms
- **Relations:** Attachment of components’ ports to connectors’ roles (interfaces with protocols)
- **Properties:** specify information for construction & analysis
  - Quality attributes (to help analysis)
  - Others, depending on style

What Are C&C Views Used For?

- **Construction:** Define how the system will appear at run time, and therefore determines what kind of behavior must be built in. Also, pathways of interaction, and communication mechanisms.
- **Analysis:** Runtime properties of a system, such as availability, performance, aspects of security, reliability, ...

C&C Structural Concepts

Example C&C Style: Pipes and Filters

Data flow

Example C&C Style: Event-oriented Systems

Event announcement

A Taxonomy of C&C Architectural Styles

<table>
<thead>
<tr>
<th>Data Flow</th>
<th>Data-oriented repository</th>
</tr>
</thead>
<tbody>
<tr>
<td>Batch sequential</td>
<td>Transactional databases</td>
</tr>
<tr>
<td>Dataflow network</td>
<td>True client-server</td>
</tr>
<tr>
<td>pipes&amp;filters</td>
<td>Blackboard</td>
</tr>
<tr>
<td>acyclic, fanout,</td>
<td>Modern compiler</td>
</tr>
<tr>
<td>pipeline, Unix</td>
<td></td>
</tr>
<tr>
<td>Closed loop control</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Call-and-return</th>
<th>Compound documents</th>
</tr>
</thead>
<tbody>
<tr>
<td>Main program/subroutines</td>
<td>Hypertext</td>
</tr>
<tr>
<td>Information hiding</td>
<td>Fortran COMMON</td>
</tr>
<tr>
<td>Objects, naive client-server</td>
<td>LW processes</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Interacting processes</th>
<th>Hierarchical</th>
</tr>
</thead>
<tbody>
<tr>
<td>Communicating processes</td>
<td>Tiers</td>
</tr>
<tr>
<td>LW processes, distrb objects</td>
<td>Interpreter</td>
</tr>
<tr>
<td>Event systems</td>
<td>N-tiered client-server systems</td>
</tr>
<tr>
<td>implicit invoc., pure events</td>
<td></td>
</tr>
</tbody>
</table>
Allocation Viewtype

- **Elements:**
  - Software elements (as defined in module or C&C styles)
  - Environment elements

- **Relations:** allocated-to

---

Styles of the allocation viewtype

- **Deployment style**
  - Allocates software elements to processing and communication nodes
  - Properties include those necessary to calculate (and achieve) performance, availability

- **Implementation style**
  - Allocates software elements to structures in the development environment’s file systems
  - Properties include files and capacities

- **Work assignment style**
  - Allocates software elements to organizational work units
  - Properties include skill sets

---

Allocation viewtype: Software elements and environment elements

Deployment Style: Example

C&C View of Test Parsing System

Module Decomposition Style
Implementation Style for the same system

- Systems often use a combination of styles
  - Example: tiers + repository + events
- But we should not ignore the pure versions
  - Need to understand the basic conceptual building blocks
- Moreover, in general it’s a bad idea to mix styles from different viewtypes
  - Leads to confusion
  - Example: layers and tiers

Specialized Architectural Styles

- There is a spectrum of architectural styles
  - Some are generic; others are more domain-specific and specialized.
  - Specialization supports analyses, code reuse, tools

Summary

- Views allow us to manage what we say about an architecture
- Viewtypes determine the category of view
- Three primary viewtypes
  - Module, C&C, Allocation
- Each viewtype has many styles
  - Module: Decomposition, Layered, …
  - C&C: Pipe & Filter, Client-server, …
  - Allocation: Deployment, …

Next Two Lectures

- We will consider how we can formalize architectural structure, focusing component and connector views and styles
- We then consider some of the practical issues of supporting architectural design: tools, implementation conformance, and dynamic self-healing systems.

Readings

- Lecture 1
- Lecture 2
Readings (continued)

- Lecture 3

Book References

- Software Architecture in Practice, 2nd Edition
- Software Architecture: Perspectives on an Emerging Discipline, Shaw, Garlan
  Prentice-Hall, 1996.