Software Architecture and the UML

Grady Booch

Architecting a dog house

Can be built by one person
Requires
  Minimal modeling
  Simple process
  Simple tools
Architecting a house

Built most efficiently and timely by a team
Requires
  Modeling
  Well-defined process
  Power tools

Architecting a high rise
Early architecture

**Progress**
- Limited knowledge of theory

Modern architecture

**Progress**
- Advances in materials
- Advances in analysis

**Scale**
- 5 times the span of the Pantheon
- 3 times the height of Cheops
Modeling a house

Movements in civil architecture

- Bronze age/Egyptian (Imhotep)
- Grecian/Roman (Vitruvius)
- Byzantine/Romanesque
- Gothic
- Mannerism (Michelangelo, Palladio)
- Baroque
- Engineering/Rational/National/Romantic
- Art nouveau
- Modern movement (Wright, LeCorbusier)

Progress
- Imitation of previous efforts
- Learning from failure
- Integration of other forces
- Experimentation
Kinds of civil architecture

- Community
  - houses, flats and apartments, gardens, education, hospitals, religion

- Commerce
  - shops and stores, restaurants, hotels, office buildings, banks, airports

- Industry
  - industrial buildings, laboratories, farm buildings

- Leisure
  - sport, theaters and cinemas, museums

Forces in civil architecture

- Compression
- Tension

Kinds of loads
- Dead loads
- Live loads
- Dynamic loads

Avoiding failure
- Safety factors
- Redundancy
- Equilibrium

Any time you depart from established practice, make ten times the effort, ten times the investigation. Especially on a very large project.

- LeMessuier
Shearing layers of change

Dimensions of software complexity

Higher technical complexity
- Embedded, real-time, distributed, fault-tolerant
- Custom, unprecedented, architecture reengineering
- High performance

Lower technical complexity
- Mostly 4GL, or component-based
- Application reengineering
- Interactive performance

Lower management complexity
- Small scale
- Formal
- Single stakeholder
- "Products"

Higher management complexity
- Large scale
- Contractual
- Many stakeholders
- "Projects"
Forces in Software

- Functionality
- Cost
- Compatibility
- Fail safe
- Fault tolerance
- Throughput
- Technology churn
- Resilience

The challenge over the next 20 years will not be speed or cost or performance; it will be a question of complexity.

Our enemy is complexity, and it’s our goal to kill it.

Jan Racz

The domain of architecting

The “what”
- Architecture
- Architecture Qualities
- Architecture Representation

The “who”
- Architect
- Skills
- Stakeholders

The “how”
- Organization
- Process

The “why”
- System Features
- SW Requirements
- System Quality Attributes

Satisfies
Constrain
Follows
defines role
We all know that ...

Architecture and design are the same thing
Architecture and infrastructure are the same thing
<my favorite technology> is the architecture
A good architecture is the work of a single architect
Architecture is flat, one blueprint is enough
Architecture is just structure
System architecture precedes software architecture
Architecture cannot be measured and validated
Architecture is a Science
Architecture is an Art

Architecture defined (again)

Architecture n (1555) 1: the art of science of building, specifically, the art or practice of designing and building structures and esp. habitable ones 2 a: formation or construction as or as if as the result of conscious act <the ~ of the garden> b: a unifying or coherent form or structure <the novel lacks ~>
Architecture defined (yet again)

- Software architecture encompasses the set of significant decisions about the organization of a software system
  - selection of the structural elements and their interfaces by which a system is composed
  - behavior as specified in collaborations among those elements
  - composition of these structural and behavioral elements into larger subsystem
  - architectural style that guides this organization

Architecture defined (continued)

- Software architecture also involves
  - usage
  - functionality
  - performance
  - resilience
  - reuse
  - comprehensibility
  - economic and technology constraints and tradeoffs
  - aesthetic concerns
**Architectural style**

- An architecture style defines a family of systems in terms of a pattern of structural organization.

- An architectural style defines
  - a vocabulary of components and connector types
  - a set of constraints on how they can be combined
  - one or more semantic models that specify how a system’s overall properties can be determined from the properties of its parts

**Architecture metamodel**

![Architecture Metamodel Diagram]
Models

- Models are the language of the designer, in many disciplines
- Models are representations of the system to-be-built or as-built
- Models are a vehicle for communications with various stakeholders
- Visual models, blueprints
- Scale
- Models allow reasoning about some characteristic of the real system

Many stakeholders, many views

- Architecture is many things to many different interested parties
  - end-user
  - customer
  - project manager
  - system engineer
  - developer
  - architect
  - maintainer
  - other developers
- Multidimensional reality
- Multiple stakeholders
  - multiple views, multiple blueprints
Architectural view

- An architectural view is a simplified description (an abstraction) of a system from a particular perspective or vantage point, covering particular concerns, and omitting entities that are not relevant to this perspective.

Architecturally significant elements

- Not all design is architecture.
- Main “business” classes
- Important mechanisms
- Processors and processes
- Layers and subsystems
- Architectural views = slices through models
Characteristics of a Good Architecture

- Resilient
- Simple
- Approachable
- Clear separation of concerns
- Balanced distribution of responsibilities
- Balances economic and technology constraints

Representing System Architecture
### How many views?

- Simplified models to fit the context
- Not all systems require all views:
  - Single processor: drop deployment view
  - Single process: drop process view
  - Very Small program: drop implementation view
- Adding views:
  - Data view, security view
The Value of the UML

- Is an open standard
- Supports the entire software development lifecycle
- Supports diverse applications areas
- Is based on experience and needs of the user community
- Supported by many tools

Creating the UML

- OMG Acceptance, Nov 1997
- Final submission to OMG, Sep '97
- First submission to OMG, Jan '97
- UML partners
- Web - June '96
- OOPSLA '95
- Unified Method 0.8
- Other methods
- Booch method
- OMT
- OOSE
UML Partners

- Rational Software Corporation
- Hewlett-Packard
- I-Logix
- IBM
- ICON Computing
- Intellicorp
- MCI Systemhouse
- Microsoft
- ObjecTime
- Oracle
- Platinum Technology
- Taskon
- Texas Instruments/Sterling Software
- Unisys

Contributions to the UML

- Meyer
  - Before and after conditions
- Booch
  - Booch method
- Rumbaugh
  - CMT
- Jacobson
  -OOSE
- Shlaer - Mellor
  - Object lifecycles
- Odell
  - Classification
- Harel
  - Statecharts
- Gamma, et al
  - Frameworks and patterns
- HP Fusion
  - Operation descriptions and message numbering
- Embley
  - Singleton classes and high-level view
- Wirfs-Brock
  - Responsibilities
Overview of the UML

- The UML is a language for
  - visualizing
  - specifying
  - constructing
  - documenting

the artifacts of a software-intensive system

Overview of the UML

- Modeling elements
- Relationships
- Extensibility Mechanisms
- Diagrams
Modeling Elements

- **Structural elements**
  - class, interface, collaboration, use case, active class, component, node

- **Behavioral elements**
  - interaction, state machine

- **Grouping elements**
  - package, subsystem

- **Other elements**
  - note

Relationships

- **Dependency**
- **Association**
- **Generalization**
- **Realization**
Extensibility Mechanisms

- **Stereotype**
- **Tagged value**
- **Constraint**

```
<container>
  ActionQueue
  version = 3;
  add(a : Action)
  remove(n : Integer)
  <query>
    length() : Integer
  </query>
  <helper functions> reorder()
```

---

Models, Views, and Diagrams

A **model** is a complete description of a system from a particular perspective.

- **Sequence Diagrams**
- **Use Case Diagrams**
- **Class Diagrams**
- **Object Diagrams**
- **Component Diagrams**
- **Deployment Diagrams**
- **Collaboration Diagrams**
- **Statechart Diagrams**
- **Activity Diagrams**
Diagrams

- A diagram is a view into a model
  - Presented from the aspect of a particular stakeholder
  - Provides a partial representation of the system
  - Is semantically consistent with other views

- In the UML, there are nine standard diagrams
  - Static views: use case, class, object, component, deployment
  - Dynamic views: sequence, collaboration, statechart, activity

Use Case Diagram

- Captures system functionality as seen by users
**Use Case Diagram**

- Captures system functionality as seen by users
- Built in early stages of development
- Purpose
  - Specify the context of a system
  - Capture the requirements of a system
  - Validate a system’s architecture
  - Drive implementation and generate test cases
- Developed by analysts and domain experts

**Class Diagram**

- Captures the vocabulary of a system
Class Diagram

- Captures the vocabulary of a system
- Built and refined throughout development
- Purpose
  - Name and model concepts in the system
  - Specify collaborations
  - Specify logical database schemas
- Developed by analysts, designers, and implementers

Object Diagram

- Captures instances and links
Object Diagram

- Shows instances and links
- Built during analysis and design
- Purpose
  - Illustrate data/object structures
  - Specify snapshots
- Developed by analysts, designers, and implementers

Component Diagram

- Captures the physical structure of the implementation
Component Diagram

- Captures the physical structure of the implementation
- Built as part of architectural specification
- Purpose
  - Organize source code
  - Construct an executable release
  - Specify a physical database
- Developed by architects and programmers

Deployment Diagram

- Captures the topology of a system’s hardware
Deployment Diagram

- Captures the topology of a system’s hardware
- Built as part of architectural specification
- Purpose
  - Specify the distribution of components
  - Identify performance bottlenecks
- Developed by architects, networking engineers, and system engineers

Sequence Diagram

- Captures dynamic behavior (time-oriented)
Sequence Diagram

- Captures dynamic behavior (time-oriented)
- Purpose
  - Model flow of control
  - Illustrate typical scenarios

Collaboration Diagram

- Captures dynamic behavior (message-oriented)

```mermaid
sequenceDiagram
    Client -> DB: create
    DB: create
    Client -> DB: setActions(a, d, o)
    DB: setActions(a, d, o)
    Client -> DB: destroy
    DB: destroy
```
Collaboration Diagram

- Captures dynamic behavior (message-oriented)
- Purpose
  - Model flow of control
  - Illustrate coordination of object structure and control

Statechart Diagram

- Captures dynamic behavior (event-oriented)
Statechart Diagram

- Captures dynamic behavior (event-oriented)
- Purpose
  - Model object lifecycle
  - Model reactive objects (user interfaces, devices, etc.)

Activity Diagram

- Captures dynamic behavior (activity-oriented)
Activity Diagram

- Captures dynamic behavior (activity-oriented)
- Purpose
  - Model business workflows
  - Model operations

Architecture and the UML

[Diagram of Architecture and the UML]

- Design View
- Implementation View
- Process View
- Deployment View
- Use Case View
- Use cases
- Components
- Active classes
- Organization
- Package, subsystem
- Dynamics
- Interaction
- State machine
Software engineering process

A set of partially ordered steps intended to reach a goal. In software engineering the goal is to build a software product or to enhance an existing one.

- Architectural process
  - Sequence of activities that lead to the production of architectural artifacts:
    - A software architecture description
    - An architectural prototype

Rational Unified Process

- Iterative
- Architecture-centric
- Use-case driven
- Risk confronting
Focus over time

- Discovery
- Invention
- Implementation

Focus

Key concepts

- Phase, Iterations
- Process Workflows
  - Activity, steps
- Artifacts
  - models
  - reports, documents
- Worker: Architect

When does architecture happen?
What does happen?
What is produced?
Who does it?
Lifecycle Phases

<table>
<thead>
<tr>
<th>Inception</th>
<th>Elaboration</th>
<th>Construction</th>
<th>Transition</th>
</tr>
</thead>
</table>

Inception: Define the scope of the project and develop business case
Elaboration: Plan project, specify features, and baseline the architecture
Construction: Build the product
Transition: Transition the product to its users

Major Milestones

<table>
<thead>
<tr>
<th>Inception</th>
<th>Elaboration</th>
<th>Construction</th>
<th>Transition</th>
</tr>
</thead>
</table>

Vision
Baseline Architecture
Initial Capability
Product Release
Phases and Iterations

<table>
<thead>
<tr>
<th>Inception</th>
<th>Elaboration</th>
<th>Construction</th>
<th>Transition</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

An iteration is a sequence of activities with an established plan and evaluation criteria, resulting in an executable release.

Architecture-Centric

- Models are vehicles for visualizing, specifying, constructing, and documenting architecture.
- The Unified Process prescribes the successive refinement of an executable architecture.
Unified Process structure

Process Workflows
- Business Modeling
- Requirements
- Analysis & Design
- Implementation
- Test
- Deployment

Supporting Workflows
- Configuration Mgmt
- Management
- Environment

Phases
- Inception
- Elaboration
- Construction
- Transition

Architecture and Iterations
- Use case Model
- Design Model
- Implementation Model
- Deployment Model
- Test Model

Iterations
- Preliminary Iteration(s)
- Iter. #1
- Iter. #2
- Iter. #m
- Iter. #m+1
- Iter. #n
- Iter. #n+1
- Iter. #n+2
- Iter. #n+3

Content
Architectural design

- Identify, select, and validate “architecturally significant” elements
- Not everything is architecture
  - Main “business” classes
  - Important mechanisms
  - Processors and processes
  - Layers and subsystems
  - Interfaces
- Produce a Software Architecture Document

Architectural design workflow

- Select scenarios: criticality and risk
- Identify main classes and their responsibility
- Distribute behavior on classes
- Structure in subsystems, layers, define interfaces
- Define distribution and concurrency
- Implement architectural prototype
- Derive tests from use cases
- Evaluate architecture

Iterate
Sources of architecture

Theft  **Method**  Intuition

Classical system

Unprecedented system

Patterns

- A pattern is a solution to a problem in a context
- A pattern codifies specific knowledge collected from experience in a domain
- All well-structured systems are full of patterns
  - Idioms
  - Design patterns
  - Architectural patterns
Mechanisms

- Screws
- Keys
- Rivets
- Bearings
- Pins, axles, shafts
- Couplings
- Ropes, belts, and chains
- Friction wheels
- Toothed wheels
- Flywheels
- Levers and connecting rods
- Click wheels and gears
- Ratchets

- Brakes
- Pipes
- Valves
- Springs
- Cranks and rods
- Cams
- Pulleys
- Engaging gears

Design patterns

- Creational patterns
  - Abstract factory
  - Prototype
- Structural patterns
  - Adapter
  - Bridge
  - Proxy
- Behavioral patterns
  - Chain of responsibility
  - Mediator
  - Visitor
- Mechanisms are the soul of an architecture
Modeling a design pattern

Modeling a design pattern (cont.)
Modeling a design pattern (cont.)

Architectural patterns

- Distributed
- Event-driven
- Frame-based
- Batch
- Pipes and filters
- Repository-centric
- Blackboard
- Interpreter
- Rule-based

- Layered
- MVC
- IR-centric
- Subsumption
- Disposable
Complex business system

Logical application architecture
Physical application architecture

Complex Internet system
Who are the architects?

- Experience
  - software development
  - domain
- Pro-active, goal oriented
- Leadership, authority
- Architecture team
  - balance

Architect

- Not just a top level designer
  - Need to ensure feasibility
- Not the project manager
  - But “joined at the hip”
- Not a technology expert
  - Purpose of the system, “fit”,
- Not a lone scientist
  - Communicator
Software architecture team charter

- Defining the architecture of the software
- Maintaining the architectural integrity of the software
- Assessing technical risks related to the software design
- Proposing the order and contents of the successive iterations
- Consulting services
- Assisting marketing for future product definition
- Facilitating communications between project teams

Architecture is making decisions

The life of a software architect is a long (and sometimes painful) succession of suboptimal decisions made partly in the dark.
Futures

- ADL: Architecture Description Languages
  - UML, UniCon, LILEAnna, P++, LEAP, Wright, μRapid
- Standardization of concepts
  - IEEE Working Group on Architecture
  - INCOSE Working Group on System Architecture
- Systematic capture of architectural patterns

References (Architecture)

- Frank Buschmann, Régine Meunier, Hans Rohnert, Peter Sommerlad, and Michael Stahl, Pattern-Oriented Software Architecture - A System of Patterns, Wiley and Sons, 1996.
- Eric Gamma, John Vlissides, Richard Helm, Ralph Johnson, Design Patterns, Addison-Wesley 1995.
## References (Architecture)

- The World-wide Institute of Software Architects
  - [http://www.wwisa.org](http://www.wwisa.org)

## References (UML)

References (Process)

  - http://www.rational.com/support/techpapers/devprcs/
- The Software Program Manager’s Network
  - http://www.spmn.com